Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Design and Evaluation of Algorithms for Dynamic Resource Management on SDN/NFV Networks

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Andrés Sánchez Ramos

ADVISOR: Cristina Cervelló-Pastor, Sebastià Sallent

DATE: May, 2nd 2018

Title: Design and Evaluation of Algorithms for Dynamic Resource Management on SDN/NFV Networks

Author: Andrés Sánchez Ramos

Advisor: Cristina Cervelló-Pastor, Sebastià Sallent

Date: May, 2nd 2018

Abstract

Internet service providers and telecommunications operators have traditionally deployed networks installing various hardware appliances linked together using physical static links, these devices are configured through a broad set of tools to provide services. The aforementioned devices tend to be highly specialized and customized, usually having their operative system tightly coupled to the underlying hardware. Communication among devices is achieved through the implementation of strict protocols to achieve reliability.

The aforementioned approach altogether with the exponential demand increase experienced in modern networks have proved that these networks are expensive to operate, are difficult to scale, require highly specialized maintenance and are not flexible. This has brought difficulties to service providers, since this increase in the demand of quality in their services is not accompanied by an increase in the service fees.

Network Function Virtualization (NFV) was introduced by service operators as a new paradigm proposing to address these difficulties, looking to obtain similar benefits as those obtained in Information technology (IT) environments leveraging server virtualization. In the present document a brief review is done of the state of the art technologies involved in Service Function Chaining (SFC) in NFV environments, including a high-level description of the main components involved in open-source NFV and a detailed description of the open-source orchestrator Open Source MANO (OSM). Finally, an introduction to a framework for providing resource optimization assignment algorithms in orchestrators is done, providing preliminary results. Although the developed components do not achieve the requirements for acceptance into a production environment they provide a starting point for integration into production orchestration environments.

CONTENTS

INTRODUCTION

Modern carrier networks have grown to be complex and difficult to operate; they are usually comprised by proprietary hardware appliances deployed in static layouts. This conditions cause traditional networks to be non-flexible and to have an expensive lifecycle. Introducing new services is often a difficult task requiring the procurement of new equipment, a re-design of hardware connections and the availability of physical space and power capacity.

Network Functions Virtualization (NFV) and Software-Defined Networking (SDN) are new paradigms that propose to change the traditional network service deployment model. They will enable service providers to leverage the benefits that virtualization has brought to Information Technology (IT) services. These technologies are based on an open-source and decoupled model that will help address the lack of flexibility in service provider's networks. The final objective NFV and SDN have is to host network functions on commodity computing and networking hardware, thus escaping from the traditional hardware-based appliance model.

Traditional network services are made of several hardware appliances connected in such a way that traffic traverses them in a logical order and provides static forwarding paths. This concatenation of Network Functions (NF) is defined as a Service Function Chain (SFC). The introduction of NFV and SDN will provide a flexible provisioning scheme for the creation of service chains, providing the ability to deploy tailored services to customers. This is done by the concatenation of several Virtual Network Functions (VNF) acting as building blocks chained together and providing the ability to be modified dynamically.

The aim of this thesis is to perform a study of the state of the art technologies involved in service chaining within Network Function Virtualization, with a particular focus on service orchestrators in NFV architecture.

The project's methodology involves research activities, implementation activities and development. First a review on NFV orchestrator platforms was conducted, implementing some of them in laboratory scenarios and finally introducing the bases for developing a resource assignment optimization module that can be included in them.

The document has been divided into the following chapter to describe the project development phases:

1. Introduction to NFV and SFC.
2. State of the Art in Open Source NFV.
3. OSMANO.
4. Resource Assignment Optimization.
5. Resource Assignment Framework.
6. Conclusions

CHAPTER 1. INTRODUCTION TO NFV AND SFC

This chapter provides an introduction to the core components which comprehend a Network Function Virtualization (NFV) environment, analyze their interactions, as well as describe the drivers and limitations that are pushing the carrier provider industry towards a virtualized service architecture

1.1. Market Drivers

Networks operators have traditionally deployed network services as physical proprietary devices which are inserted into the network. These service modelling has made the deployment and operation process complex, since it requires proper physical infrastructure, skilled professionals, high energy consumption and large investments.

Modern IT trends are pressuring carrier service providers to be able to provide flexible and dynamic services, as well as providing higher transmission capacities while maintaining static pricing schemes because of fierce market competition. The unification of IP communications as the preferred method to transmitting voice, video and data; together with the common use of smartphones has dramatically increased the capacity demanded to network operators by end users.

Network Function Virtualization has been pointed as a way to address some of these challenges, since it promises to bring all the benefits already experienced in IT Data Center virtualization to carrier service provider's networks. This new service architecture model will be able to reduce operational and capital expenses, provide service deployment flexibility, reduce time to market to new services, introduce dynamic scaling to network functions, achieve space and energy savings, optimize infrastructure utilization, among other proposed benefits.

In summary network function virtualization will disrupt the traditional network service design and implementation, by removing the rigidness and proprietary characteristics of traditional networking equipment. It will decouple physical network equipment from the services they provide, using general purpose servers, it will open the network function market to new segments and it will encourage innovation and openness.

1.2. Introduction to NFV

Network Function Virtualization aims to consolidate many network equipment types onto industry standard high volume servers, switches and storage, which could be located in data centers, network nodes and in end user premises. It involves the implementation of network functions in software that can run on a range of industry standard server hardware, and that can be moved to, or

instantiated in various locations in the network as required, without the need for installation of new equipment [2]. Network virtualization implies the coexistence of virtual networks on a shared physical infrastructure; a virtual network is a collection of virtual nodes and virtual links.

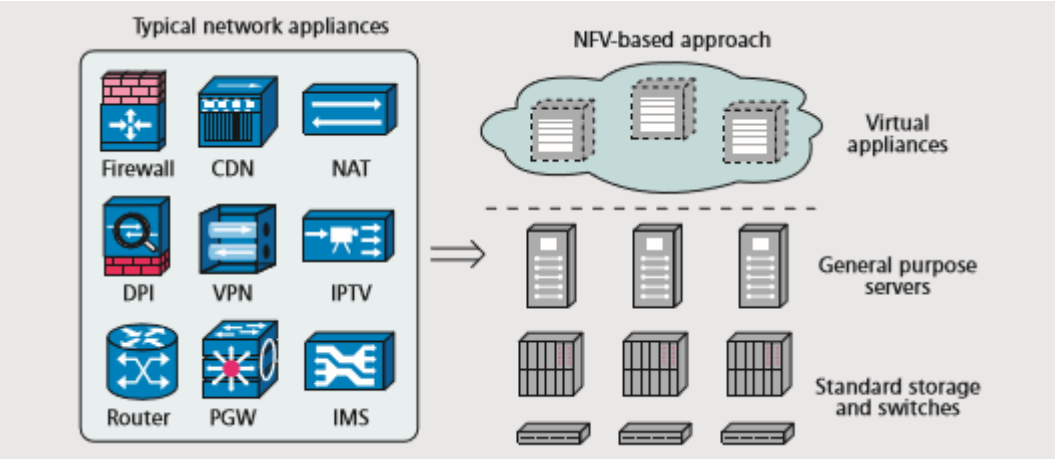

Figure 1.1 Transition from dedicated hardware appliances to software-based solutions [1].

In traditional networks the functions deployed are a combination of vendor specific hardware and software, referred to as network element. Network Function Virtualization introduces some differences in the way the network is provisioned: hardware and software are decoupled therefore their evolution is independent, the detachment of software and hardware enables the establishment of a pool of resources which can automate network function instantiation, the decoupling of network functionality into instantiable software components provides flexibility to scale the Virtual Network Function (VNF) performance according to the actual traffic for which the network operator needs to provision capacity.

The architectural framework proposed by ETSI is composed of the following elements [3]:

- *Virtual Network Function* (VNF): virtualization of a network function, examples are: serving gateway, DHCP, Evolved Packet Core.
- *Element Management*: performs management functions to one or several VNFs.
- *NFV Infrastructure* (NFVI): comprises all the hardware and software components that build the environment where the VNFs are deployed, managed and executed. This infrastructure spans across several locations, including the network providing connectivity between these points of presence. It is divided in hardware resources, virtualization layer and virtualized resources.
- *Virtualized Infrastructure Manager* (VIM): performs inventory of software and hardware components, allocates resources to virtual machines, allocates virtualization enablers, visibility into infrastructure management,

analysis of performance issues, collect fault information, collects information for capacity planning, monitoring and optimization.

- *NFV Orchestrator*: is in charge of the orchestration and management of NFV infrastructure and software resources, and realizing network services on the NFVI.
- *VNF Manager*: responsible for VNF lifecycle management (instantiation, update, query, scaling, and termination). Multiple manager might be deployed, each serving one or multiple VNFs.
- Service, VNF and Infrastructure Description: It is a data set providing information regarding VNF deployment template, VNF forwarding graph, service-related information, and NFV infrastructure information models.
- Operations and Business Support Systems (OSS/BSS): IT system for monitoring, controlling, analyzing and managing a network system.

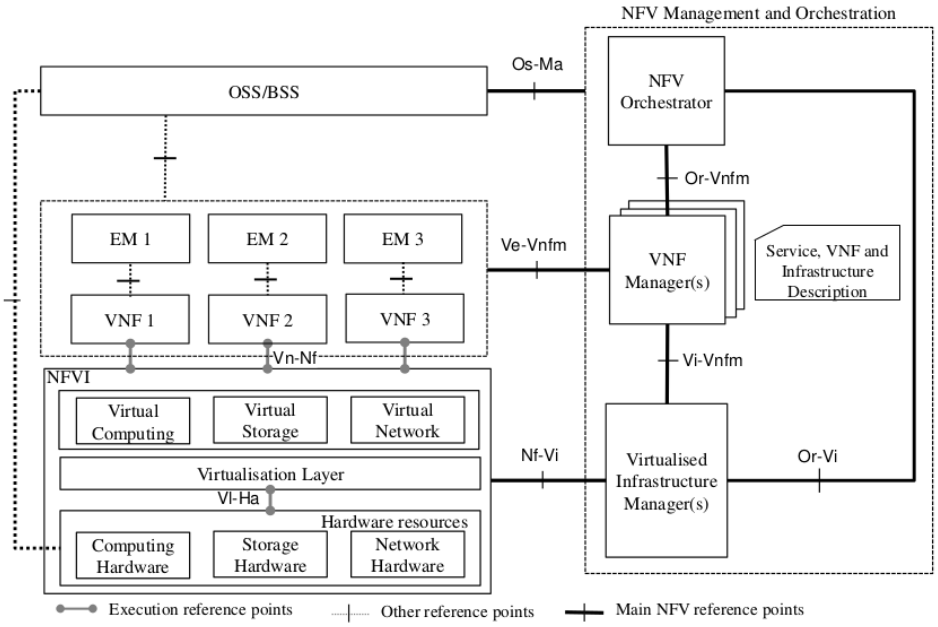

Figure 1.2. NFV reference architectural framework [3].

1.3. SDN, NFV and Cloud Computing

Network Function Virtualization holds tight relationship with Software Defined Networking and Cloud Computing concepts. The interaction between these new technologies enables the deployment of agile, automated and resource-efficient services. NFV leverages on cloud computing to build pool of physical resources which comprise the virtual infrastructure and SDN provides a centralized control plane for the overlay network that will connect the virtual network functions.

1.3.1. Software Defined Networking

Software Defined Networking is an emerging architecture that decouples the network control and forwarding functions, enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for application and network services. The OpenFlow protocol is a foundational element for building SDN solutions. [4]

SDN enables devices to be programmable via standard interfaces, this feature allows for more flexible traffic routing decisions compared to traditional methods since network administrators are able to implement forwarding rules that can match several header fields on incoming packets.

SDN architecture is comprised by a centralized controller and networking devices running a software agent; the SDN controller has northbound interfaces that are API which enable controller programming and southbound interfaces that communicate with networking devices. These devices might be a dedicated networking element but may also be a virtual switch on a standard server connecting virtual machines.

The SDN architecture is:

- Programmable: Network control is programmable since it is decoupled from forwarding functions.
- Agile: Since network control is centralized and abstracted it allows to dynamically adjust network traffic flows.
- Centrally managed: Control plane is centralized in a software based controller that maintains a complete view of the network.
- Open standard and vendor neutral: SDN is conceived to use open standards and protocols, avoiding vendor specific devices; therefore, simplifying network design and operation.

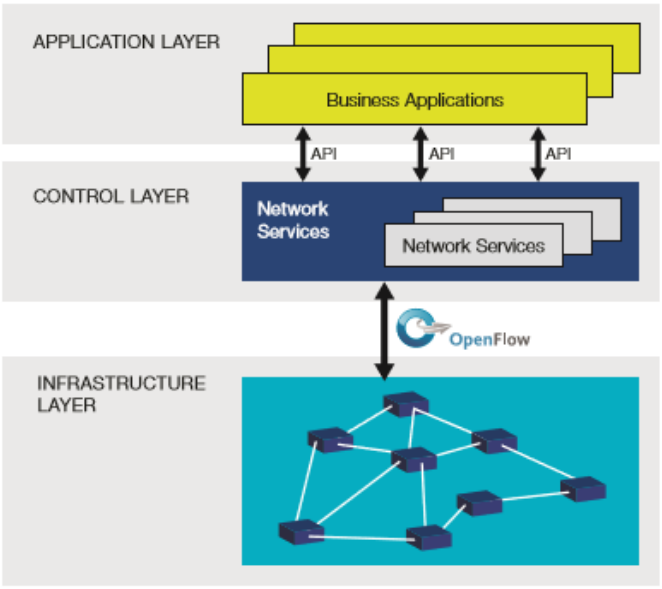

Figure 1.3. Software Defined Networking Architecture. [5]

1.3.2. Cloud Computing

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. [6]

Cloud computing is the basic building block for NFV since it provides the foundation for the operation and management of the physical infrastructure where the virtual network functions will be hosted. These components are referenced in ETSI architecture as the Network Function Virtualization Infrastructure. This model has provided numerous benefits in traditional IT data center deployments and is now a usual practice in application design; NFV proposes to obtain these advantages in carrier service provider´s networks.

Some of the characteristics of this architecture are: on demand self-service, multi-tenancy resources, broad network access, resource pooling, service elasticity, metering capabilities. The de-facto platform used for NFV deployments is Openstack which is an open source cloud operating system.

1.3.3. Relationship between NFV, SDN and Cloud Computing.

The aforementioned concepts are complimentary among themselves and they work together for the design of modern carrier service provider's data centers and point of presences. A complete NFV solution will leverage on both SDN and Cloud Computing in order to fulfill its service requirements, since virtual network functions will be hosted as guests in servers arranged as a public or private cloud and their network configuration will be done by employing software defined networking methods.

The cloud is the basic building block since it provides the pool of physical resources with the software to manage these components. These solutions are proven to offer rapid and flexible virtual machine deployment, application scalability and resource high availability; features fundamental for placing a virtual network function as a service that complies with the high demanding capabilities in operator's environment.

Software Defined Networking will perform the bridging role for the network services deployed in clouds; it is paramount for the architecture's success since it will provide dynamicity and automation in the cloud's networking service. SDN will enable the virtual machines to be properly connected throughout the cloud by pushing the required forwarding rules according to the network services blueprints, while also supporting flexibility as it can dynamically change configuration in network devices.

The synergy between the three concepts is what will enable a NFV solution to be:

- Flexible: Virtual machines will be instantiated on demand, as well as migrated between different physical sites; they can also be vertically scaled to accommodate different workloads.
- Agile: Services need to be created on demand and resource provisioning will be automated, with minimal administrative overhead.
- Cost reduction: Resource pooling will provide a more efficient use of available infrastructure, and the centralization of network intelligence will enable the use of standard network equipment. The softwarization of network appliances will also remove the dependency for hardware specialized appliances in some cases therefore giving the operators opportunity for better resource usage.
- On-demand deployments: the automation proposed by this architecture will enable tenants and operators to choose services from a pre-defined catalogue and deploying them without traditional overhead.

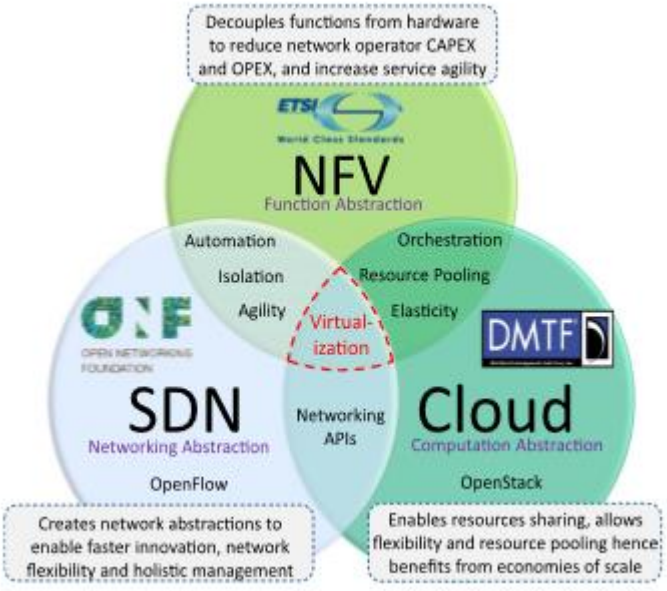

Figure 1.4. Relationship between NFV, SDN and Cloud Computing. [7]

1.4. Introduction to SFC

Traditional network layout consists of physical network devices connected through physical links; the network is logically segmented by using technologies like Virtual LAN to divide the broadcast domains. A network service is comprised by an end-to-end connection of devices which form an ordered list of network functions and traffic flows.

This model is characterized by producing static deployments that require careful planning and are very rarely modified after their initial installation, since making changes is complicated because every device in the layout might be affected by

the change in any network node. This makes it burdensome to introduce new functions or making major modifications to the network design.

1.4.1. SFC architecture

The term service function chaining (SFC) is used to describe the definition and instantiation of an ordered list of instances of service functions, and the subsequent steering of traffic flows through those service functions. [8] The aforementioned architecture is based on a workflow where packets are classified at ingress and then forwarded to be handled by the appropriate service functions defined in the chain.

The objective sought after is to flexibly classify packets as they enter the network, so they can be routed through the service chain by matching them against versatile rules based on various header fields as VLAN, source and destination addresses, source and destination MAC, ingress ports, source and destination ports, among others. This approach will enable to enforce granular policies to different kind of traffic, individual functions can be modified easily and operators will be able to implement tailored services for its customers. The IETF defines SFC core components [9]:

- *Service Function* (SF): A function that is responsible for specific treatment of received packets. A Service Function can act at various layers of a protocol stack (e.g., at the network layer or other OSI layers).
- *Service Function Forwarder* (SFF): A service function forwarder is responsible for forwarding traffic to one or more connected service functions according to information carried in the SFC encapsulation, as well as handling traffic coming back from the SF.
- *Service Function Path* (SFP): The service function path is a constrained specification of where packets assigned to a certain service function path must go.
- *SFC Encapsulation*: The SFC encapsulation provides, at a minimum, SFP identification, and is used by the SFC-aware functions, such as the SFF and SFC-aware SFs. The SFC encapsulation is not used for network packet forwarding. In addition to SFP identification, the SFC encapsulation carries metadata including data-plane context information.
- *SFC Proxy*: Removes and inserts SFC encapsulation on behalf of an SFC-unaware service function.

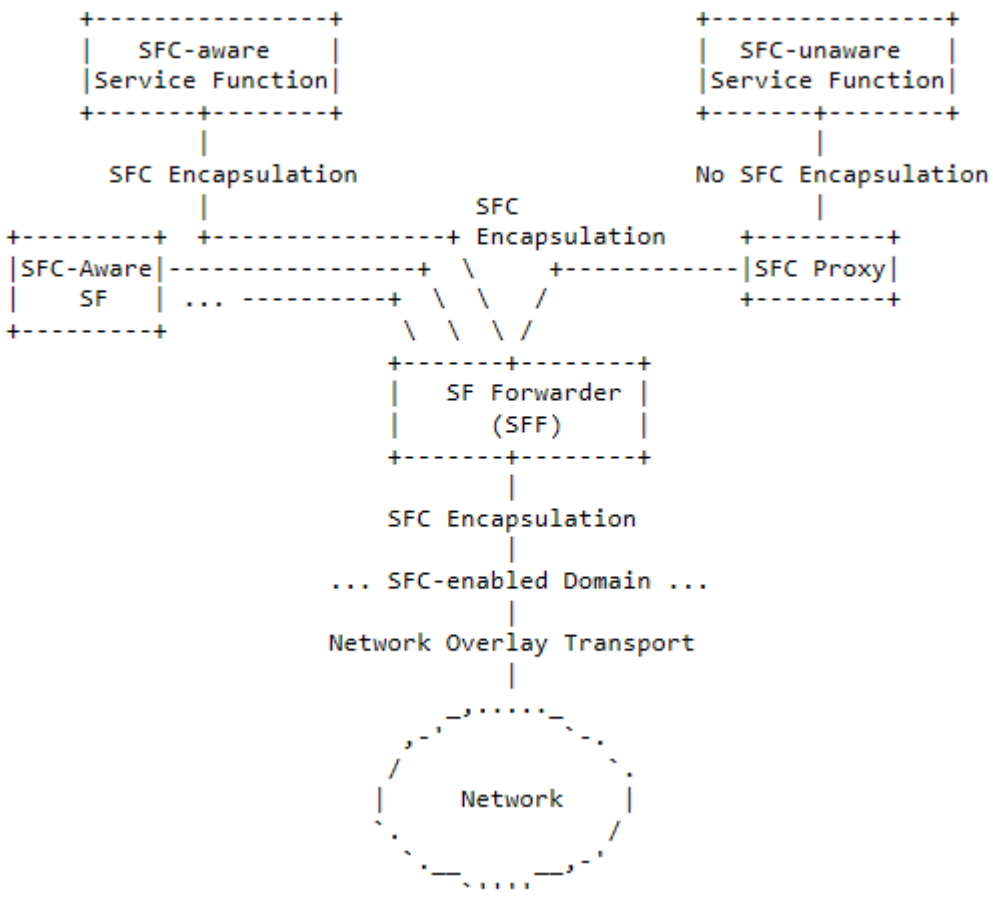

Figure 1.5. SFC architecture components. [9]

1.4.2. Network Service Headers

Network Service Headers (NSH) is a standard proposed by several manufactures, including Cisco, Microsoft and Citrix among others. It is a service plane protocol for defining dynamic chains implemented on both physical and virtual devices.

As stated in the draft [10]: A Network Service Header contains service path information and optionally metadata that are added to a packet or frame and used to create a service plane. An outer transport header is imposed, on NSH and the original packet/frame, for network forwarding.

NSH presence is indicated via protocol type or other indicator in outer encapsulation. The headers are inserted between the original packet and the outer network transport encapsulation such as MPLS, GRE or VXLAN. NSH enables traffic to be forwarded independently from connection topology, enable service chaining, carry metadata and allows for traffic to be classified and re-classified at any point of the chain.

Apart from the terms already defined by the SFC draft, NSH draft states the following terms [10]:

- *Network node*: Device that forwards packets or frames based on outer header (i.e. transport) information.
- *Network Overlay*: Logical network built on top of existing network (the underlay). Packets are encapsulated or tunnelled to create the overlay network topology.
- *Service Classifier*: Logical entity providing classification function. Classifiers may be embedded in SFC elements such as SFs or SFFs, they impose the initial NSH and sends the NSH packet to the first SFF in the path. Non-initial classification can occur as needed and can alter, or create a new service path.

The encapsulation provides for the following actions: insert or remove NSH, select service path, update NSH and service policy selection.

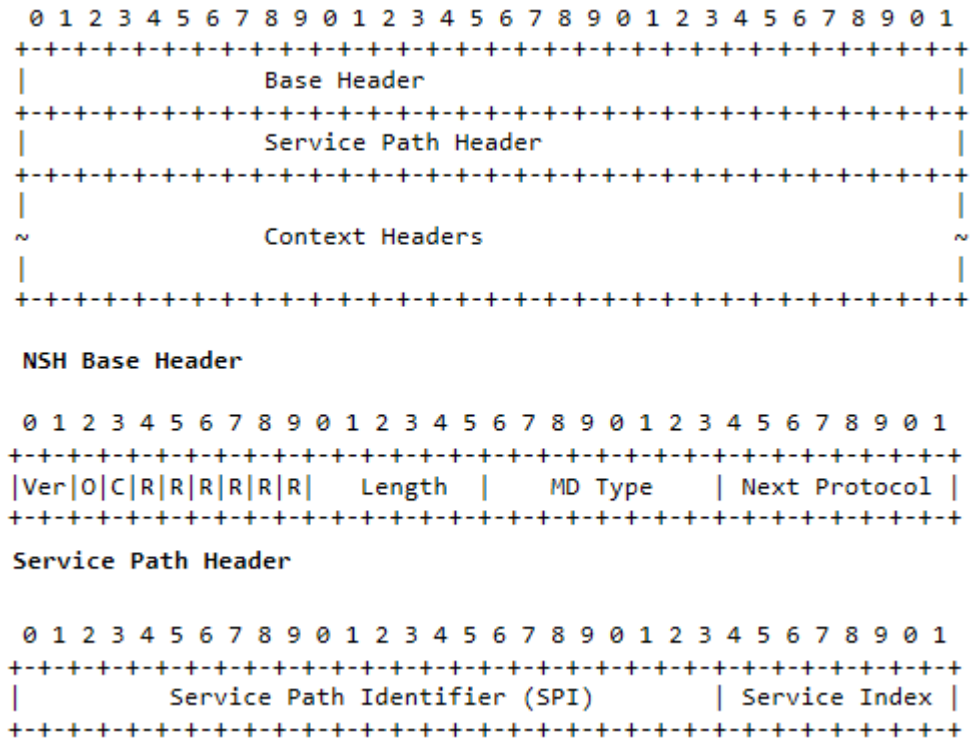

Figure 1.6. Network Service Headers. [10]

Most relevant fields in the header are the MD Type that indicates the type of metadata included in the packet, next protocol indicates the protocol type of the encapsulated data, SPI indicates the service path and Service Index the location within the SFP.

CHAPTER 2. STATE OF THE ART IN OPEN SOURCE NFV

Network Function Virtualisation was first introduced in 2012 by the publication of a white paper [11], this call was made by several telecom operators to encourage for international collaboration to start working into this subject. On the same year European Telecommunication Standard (ETSI) was chosen as the home of the Industry Specification Group (ISG) for NFV. The industry has always encouraged the use of open source software in order to achieve fast innovation and industry-wide interoperability.

The ISG works in two-year phases and are actually working on release three during 2017 and 2018 [12]. Release three is focused on developing working features of the environment and to foster an open ecosystem, some of the features being developed are: testing, DevOps and continuous integration, reliability and availability considerations, security analysis, charging and billing, among others.

NFV phase 2 was undertaken during 2015-2016, its objective was to produce normative specifications to enable end-to-end interworking of equipment and services formed a fundamental part of this phase.
The first phase took place between 2012 and 2014 and it established the architectural framework based on gathering operator's requirements, include applicable standards and to develop new technical requirements.

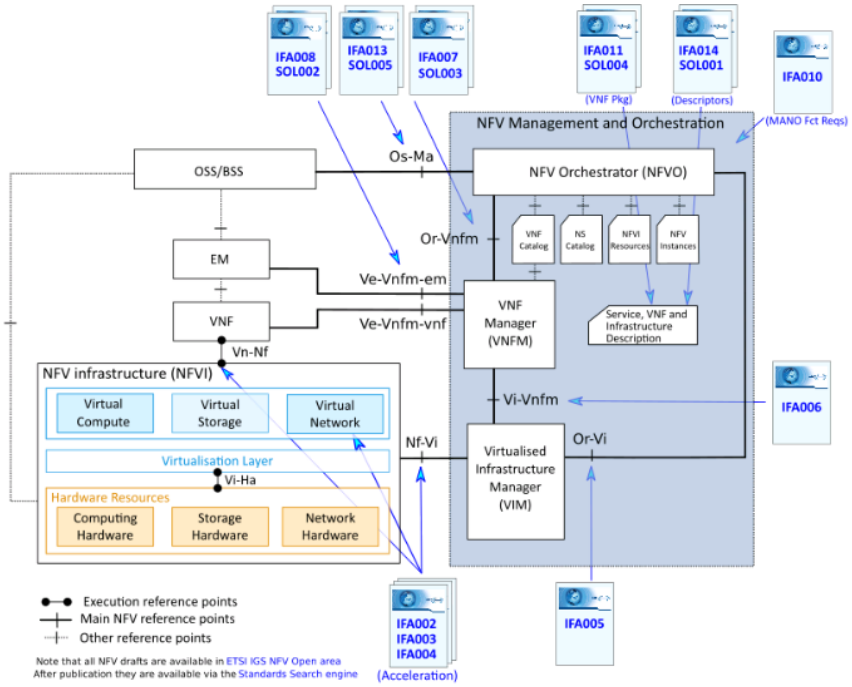

Figure 2.1. Specifications produced by ETSI ISG NFV group. [12]

2.1. Openstack

Openstack is a platform that provides cloud deployment and management, it provides services model for Infrastructure as a Service (IaaS). It is an open source virtualization platform that enables the deployment of virtual network functions using industry-standard servers. It is often used together with NFV technologies.

Large companies and start-ups like Mirantis, Cisco and Red Hat have built commercial implementations over the open source project and sell their tools and support.

2.1.1. Openstack architecture

Openstack [13] is used for creating public and private clouds; its software controls large pools of compute, storage and networking resources throughout a datacenter, managed through a dashboard or via the Openstack API. Openstack works with popular enterprise and open source technologies making it ideal for heterogeneous infrastructure. It is used as the virtual infrastructure manager in the NFV component stack.

Openstack has a six-month release cycle, the current stable release is Pike which was released in August 2018. The current developing project is Queens. It is built as loosely coupled modular components that communicate using REST based APIs. Openstack core components are [14]:

- *Nova*: The compute service which provides services to support the management of virtual machines instances.
- *Neutron*: The networking service provides various services such as IP address management, DNS, DHCP, load balancing and security groups.
- *Cinder*: Provides persistent block storage for compute instances.
- *Glance*: It is the image service that provides disk-image management services.
- *Swift*: Provides object storage services for storing and retrieving arbitrary data in the cloud.
- *Keystone*: It is the identity service that provides authentication and authorization services.
- *Horizon*: Provides a web-based interface for cloud administrators and tenants.
- *Heat*: It is an orchestration engine to launch composite cloud applications.
- *Ceilometer*: Is a data collection service that gathers event and metering data from other services.

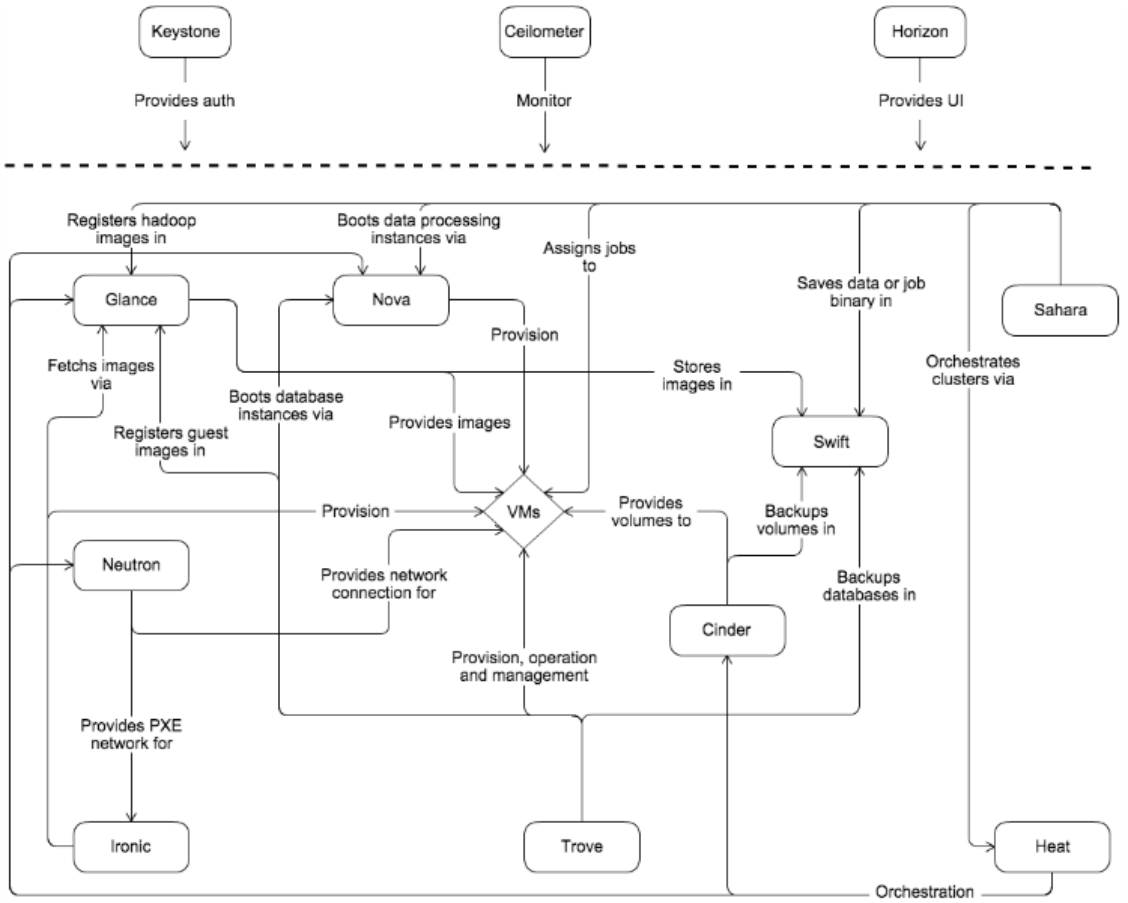

Figure 2.2. Openstack Architecture. [15]

2.1.2. Openstack for NFV

Openstack was not initially made to support telco applications, but it has evolved to support the required functions for NFV deployments. Openstack is widely deployed as the infrastructure for running VNFs by carrier service providers.

According to a survey done by Openstack [16] 84% of Carrier Service Providers have stated that Openstack is essential or important to their company's success and the same amount is actively engaged or following the project, 44% of them have "a lot" experience using the platform and 21% contribute directly by developing.

The most common use case for Openstack is to manage private cloud with 37% of carrier service providers doing so. Nearly two-thirds say that they plan to use Openstack for IoT/edge computing.

All major open source integration projects are based on Openstack as the VIM or support it among the chosen solutions, thus making it a fundamental cornerstone for VNF environment development.

2.1.3. Nova

Nova is the Openstack project that provides a way to provision compute instances, also known as virtual machines. Nova supports creating virtual machines, baremetal servers (through the use of Openstack project ironic) and has limited support for system containers. Nova runs as a set of daemons on top of existing Linux servers to provide that service [17].

Nova is composed of several processes that interact together through a messaging service; these components are [18]:

- Database: SQL database for data storage.
- API: Component that receives HTTP requests, converts commands and communicates with other components via the messaging service.
- Scheduler: Decides which host gets each instance.
- Compute: Manages communication with hypervisor and virtual machines.
- Conductor: Handles requests that need coordination (build/resize), acts as a database proxy or handles object conversions.

All services except compute are hosted on Openstack controller nodes. Compute nodes are where the virtual machines are physically hosted. Therefore, it is habitual that there are more compute nodes than any other.

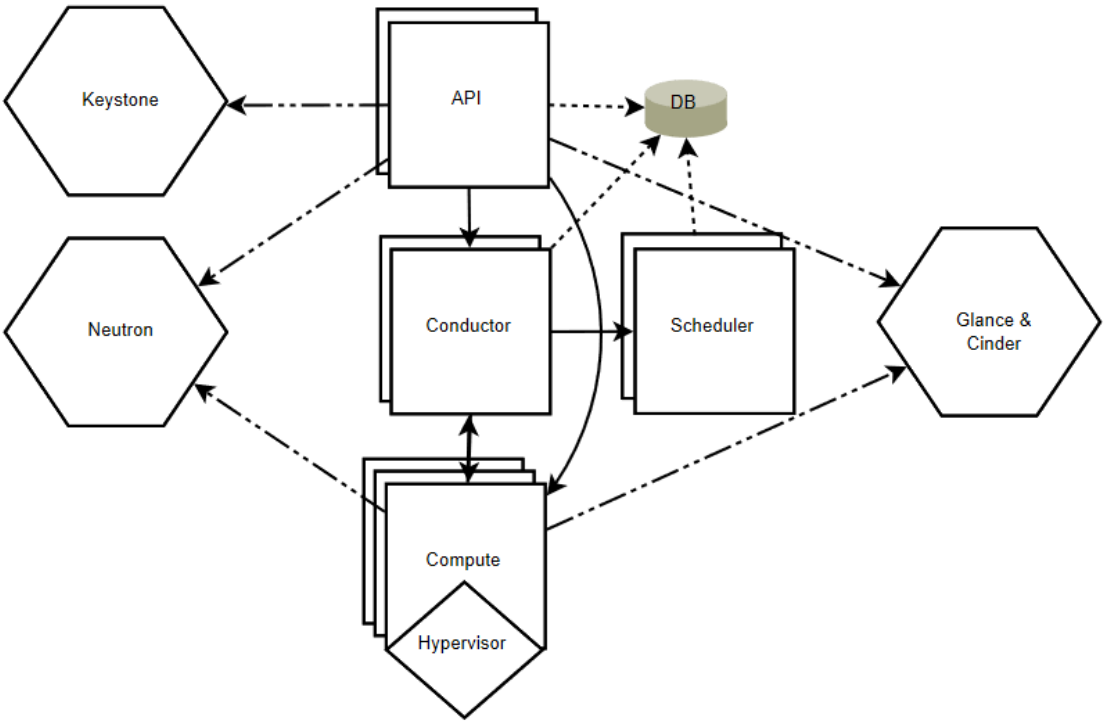

Figure 2.3. Nova System Architecture [18]

2.1.4. Neutron

Neutron is Openstack project that provides connectivity as a service between interface devices managed by other Openstack services [19]. It was originally merged with compute project, but it was divided due to its growth. It provides connectivity between virtual machines and addressing services, also supporting more complex services like routing, NAT, load balancing, firewalling and VPN.

2.1.4.1. Neutron Architecture

Neutron manages the networking for the virtual networking infrastructure and the access layer aspects of the physical networking infrastructure.

Neutron has the following components:

- *Neutron-server*: Accepts and routes API requests to the appropriate Openstack Networking plug-in for action.
- *Networking-plugins and Networking agents*: Create, delete and assign ports and subnets and provide IP addressing. Their features vary according to the technology used.
- *Messaging queue*: route information between the neutron-server and agents.

Neutron defines two types of networks, external networks also referred as provider networks which are accessible outside the Openstack installation and project networks that are software-defined and connect directly virtual machines. In order to connect virtual machines on different subnets a router between networks is required, this router operates in the same manner as a physical routerand can provide a gateway to external networks. Each plug-in that networking uses has its own concepts, all networking deployments use a core plug-in and a security group plug-in.

Most relevant networking agents are the layer 2 and layer 3 agents. The former is responsible for wiring and securing virtual interfaces while the latter provides east-west and north-south routing plus any advanced service. Layer 2 networking agent uses the Modular Layer 2 neutron plug-in which allows to simultaneously use the variety of layer 2 networking technologies found in real data centers [20]. It uses two kind of drivers: type drivers define how the network is technically realized and mechanism drivers define the mechanism to access the Openstack network. The mechanism driver correctly applies the information established by the type driver on the host devices; they can use previously define agents or interact with external devices or controllers.

Type drivers:

- Open vSwitch.
- Linux Bridge.
- SRIOV.
- MacVTap.
- L2 Population.

- Specialized: external open source or private drivers, in example OpendayLight or OpenContrail.

Mechanism drivers:

- Flat.
- VLAN.
- VXLAN.
- GRE

2.1.4.2. Neutron SFC Plugins

In order for Neutron to support Service Function Chaining capabilities to interconnect virtual machines major open source projects use the networking-SFC Openstack project that implements the software-defined networking version of policy-based routing [21].

This project provides the features for establishing service function paths, referred as port chains. Service function paths consists of a set of ports that define the sequence of service functions and a set of flow classifiers that specify the classified traffic flows entering the chain.

The port chain plug-in supports two types of implementation methods; it can be done through an OVS driver or through SDN controller drivers. These plug-ins are in charge of the service chain path rendering.

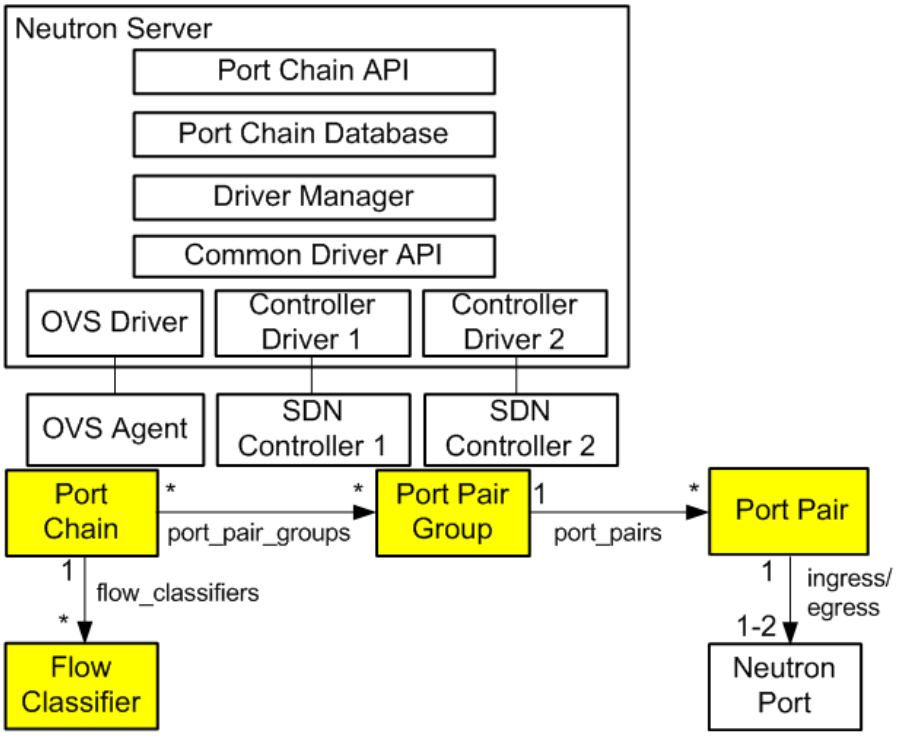

Figure 2.4. Networking-sfc architecture. [21]

Resources used for the networking configuration are:

- ID: Port chain ID.
- Project ID: Project ID
- Name: Readable name
- Description: Readable description
- Port pair groups: List of port pair group ID's
- Flow classifiers: List of flow classifier ID's
- Chain parameters: Dictionary of chain parameter.

A port chain consists of a sequence of port pair groups, a port pair group represents a hop in the port chain and a group of part pairs represents service functions with equivalent functionality. Flow classifier identifies a flow where traffic is to redirected. The chain parameters are attributes of the chain, actually supporting only MPLS; NSH support is in the roadmap. In order to support NSH implementations SDN controllers are used.

2.1.4. Heat

It is Openstack's orchestration service that automates cloud application deployment through the use of human-readable templates. These templates are text-based and provide Infrastructure as code.

Heat templates describe the infrastructure for the application, this specifies resources like servers, IP addresses, security groups, etc. They can also describe the relationship between resources and it can manage the lifecycle of the application so you can modify the template and use it to update the stack of resources. It has the following components [22]:

- Heat-api: Provides a REST API that processes requests sending them to heat-engine.
- Heat-api-cfn: Provides an AWS style query API.
- Heat-engine: Orchestrates the launch of templates and provides events back to the API consumer.

It supports HOT templates and CFN templates, HOT (Heat Orchestrator Templates) are the native format and are YAML based while CFN are CloudFormation compatible templates and are JSON based.

2.1.5. Tacker

Tacker [23] is the Openstack service for NFV orchestration. It is a VNF manager that deploys and operates virtual network functions and network services on the VIM; it can be configured to work on two types of VIMs: Openstack or Kubernetes.

Tacker is composed by three components: NFV catalog, NFV orchestrator and VNF manager. The catalog defines three types of resources: VNF descriptors, network service descriptors and VNF forwarding graph descriptors. The manager handles basic VNF life-cycle, health monitoring and auto-scaling. The orchestrator deploys network services using VNFs, does VNF placement policy, connects VNFs using SFC and checks for resources on VIMs.

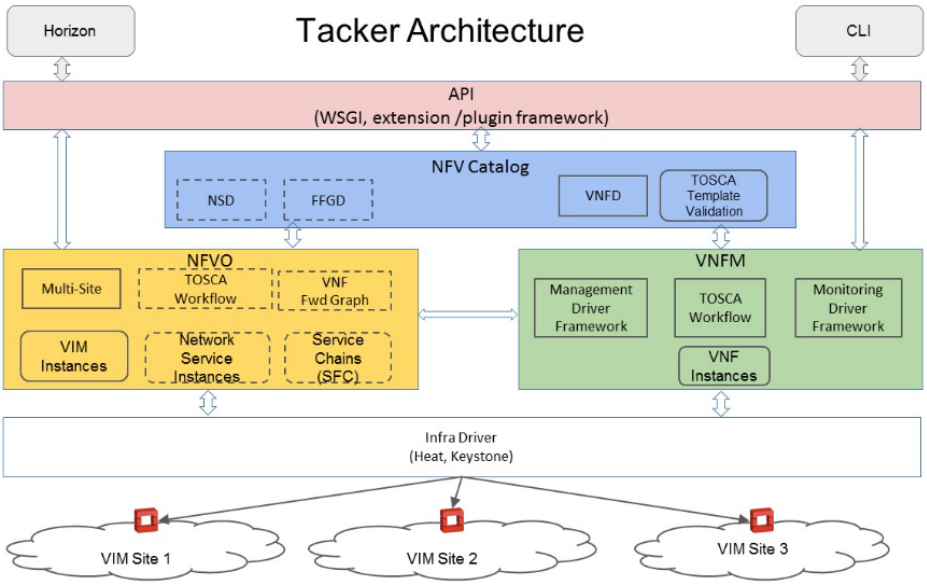

Figure 2.5. Tacker high-level architecture. [23]

The core concept for NFV applied by tacker service is the VNF forwarding graph (VNFFG), which represents the combination of NFV, SFC and the classification of traffic to flow through them. The drivers for VNFFG are a work in progress, there are two drivers to create SFCs: networking-sfc and OpenDaylight SFC; the final goal is to define a neutron driver working with networking Opendaylight to be able to create SFC for ODL via neutron. The mechanism driver that handles the classification is NetVirt. The architecture as used today in the OPNFV Euphrates integration project is:

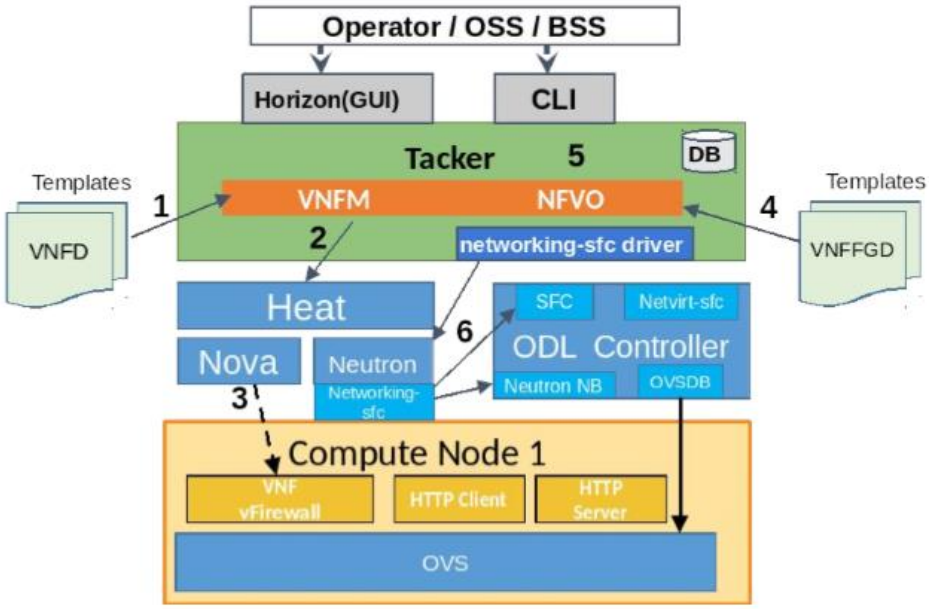

Figure 2.6. Openstack and OpenDaylight SFC implementation. [24]

2.1.6. Ceilometer

Ceilometer [25] is the data collection service that provides the ability to normalize and transform data across Openstack core components. It is part of the telemetry project and can be used for billing, resource tracking and alarming capabilities; it performs the following functions: polls metering data related to services, monitor notifications and publishes collected data.

The service consists of the following components:

- *Compute agent*: Polls for resource utilization statistics on compute nodes by interacting with the local hypervisor.
- *Central agent*: Polls for resource utilization statistics for resources not tied to instances on compute nodes by interacting with the REST APIs provided by other services.
- *Notification agent*: Consumes messages from message queues.

In order to store collected measures the service requires a database to store data, supported engines are:

- Measurements:
  - Gnocchi
- Back end for alarms:
  - MySQL.
  - PostgreSQL.
- Back end for events:
  - ElasticSearch.
  - MongoDB.
  - MySQL.
  - PostgreSQL.
  - HBase.

Supported hypervisors for collecting virtual machine information are: KVM, QEMU, LXC, UML, Hyper-V, XEN, VMware vSphere. Supported networking services are Openstack basic networking features and SDN controller meters coming from Opendaylight and Opencontrail.

2.2. OpenDaylight

OpenDaylight (ODL) [26] is a modular open platform for customizing and automating networks of any size and scale; it is intended to provide the ability to program the network. Nitrogen is the latest release of this platform which is the seventh and was released on September 2017. It is a software defined networking platform that provides centralized, programmatic control and network device monitoring.

The platform's modular design enables end users to build controllers that fulfill their particular needs since it provides the ability to incorporate any project already built or to build a new one. Controller features are packaged as Karaf containers [27].

OpenDaylight is composed by three core components: the controller, the MD-SAL and the northbound APIs. To interact with the underlying network devices, the controller does it through the Model-Driven Service Abstraction Layer (MD-SAL), where network devices and applications are represented by YANG models. MD-SAL handles southbound interactions and it supports multiple protocols as plug-ins. The controller is made by services and applications that comprise controller basic functionality like topology manager, ARP handler and device manager. Northbound APIs expose controller functionalities to applications that are not hosted on the same space as the controller.

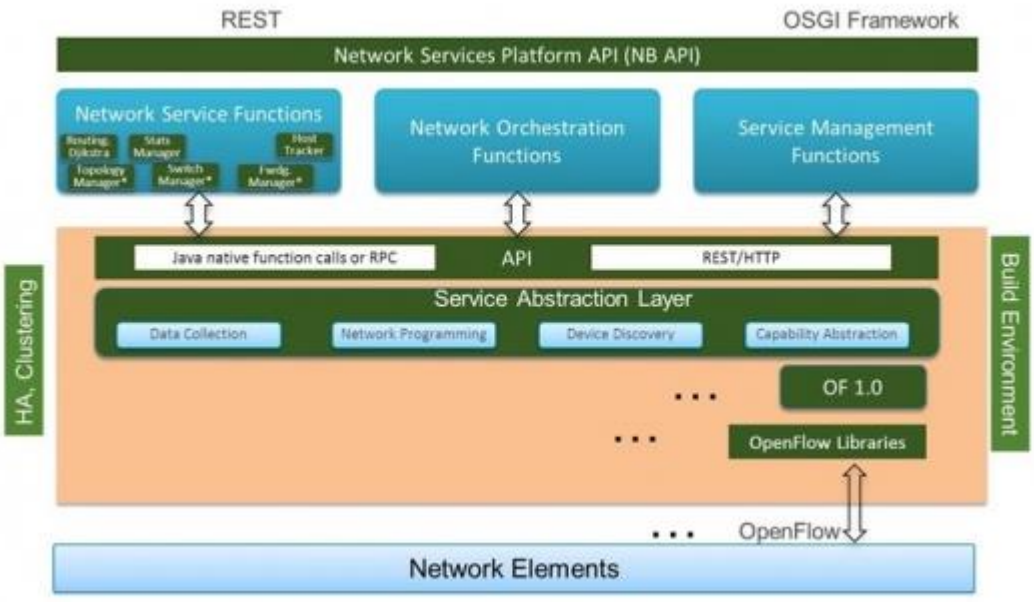

Figure 2.7. OpenDaylight Architecture. [28]

Cloud computing requires network automation to achieve dynamic management and deployment of virtual machines. OpenDaylight has included advanced support for OpenStack environments, in order to so it has developed support for the Neutron API and ML2 plug-in, and has built-in network services like network virtualization, overlay and service function chaining. This features enable OpenDaylight to successfully manage Openstack networking environments.There have already been proof of concepts that use ODL as the SDN controller for NFV scenarios.

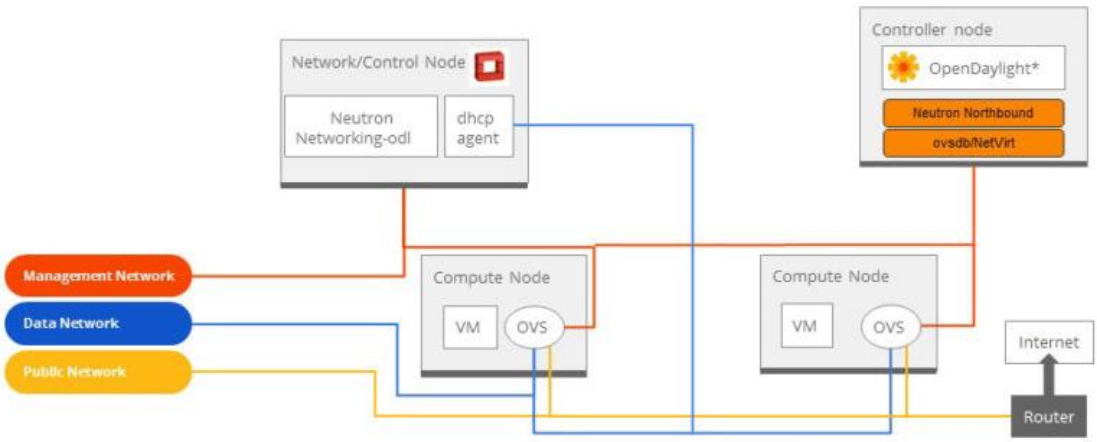

Figure 2.8. OpenStack and OpenDaylight Integration. [29]

2.3. OpenvSwitch

OpenvSwitch [30] is a multilayer virtual switch that provides traffic forwarding between virtual machines and physical networks. It is intended to be used for network automation since it provides the ability to be configured through flow rule programming and has layer 2 functionalities that enable SDN connectivity. It communicates with SDN controllers via the Openflow protocol.

It supports several linux-based virtualization technologies as Xen, KVM and VirtualBox; it can operate in UserSpace and in Kernel Space. The most important components are:

- *Ovs-vswitchd*: The daemon that implements the switch.
- *Ovsdb-server*: A lightweight database that ovs-vswitch queries to obtain its configuration.
- *Ovs-dpctl*: A tool for configuring the switch kernel module.
- *Ovs-vsctl*: utility for querying and updating the configuration of ovs-vswitchd.
- *Ovs-appctl*: A utility that sends commands to running OpenvSwitch daemons.

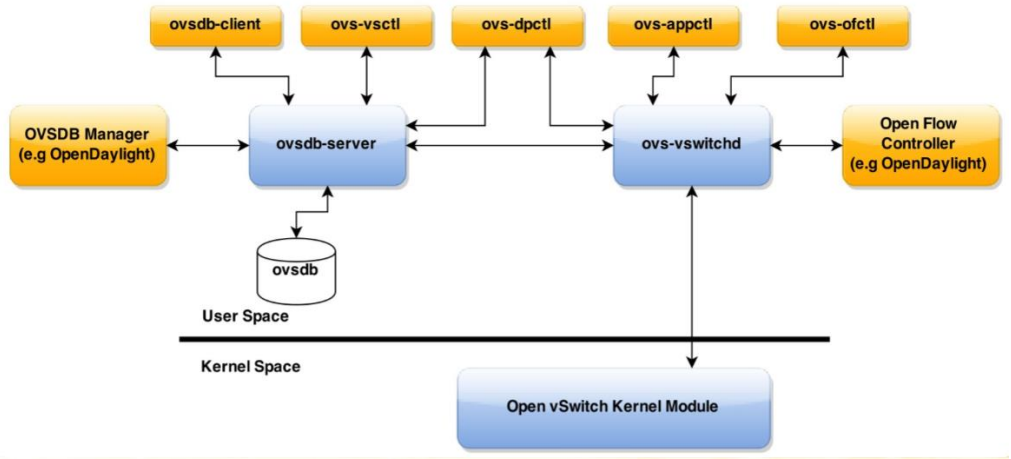

Figure 2.9. OpenvSwitch Architecture. [31]

OpenvSwitch is favored over other networking stacks because it is intended for multi-server use, supports migration of the complete network state between virtual instances, has several mechanisms that respond to networking dynamics characteristic in virtual environments, manages logical tags for traffic forwarding, support GRE tunnels and it can be implemented in both bare-metal and virtualized hosting environments.

Regarding Service Function Chaining OpenvSwitch implements the core functions needed for service traffic forwarding since it is the logical bridge that connects physical networks to virtual machines. It is the traffic classifier that adds the corresponding tags to traffic, the service function forwarder and the SFC proxy. It does this through flexible network forwarding rules achieved by flow table programming and mapping of SFC objects to SFC objects.

The classifier is implemented as ACL flow table rules, which matches traffic flows on certain header fields and push the corresponding encapsulation headers which specify the path. The service function forwarder is mapped to an OVS bridge.

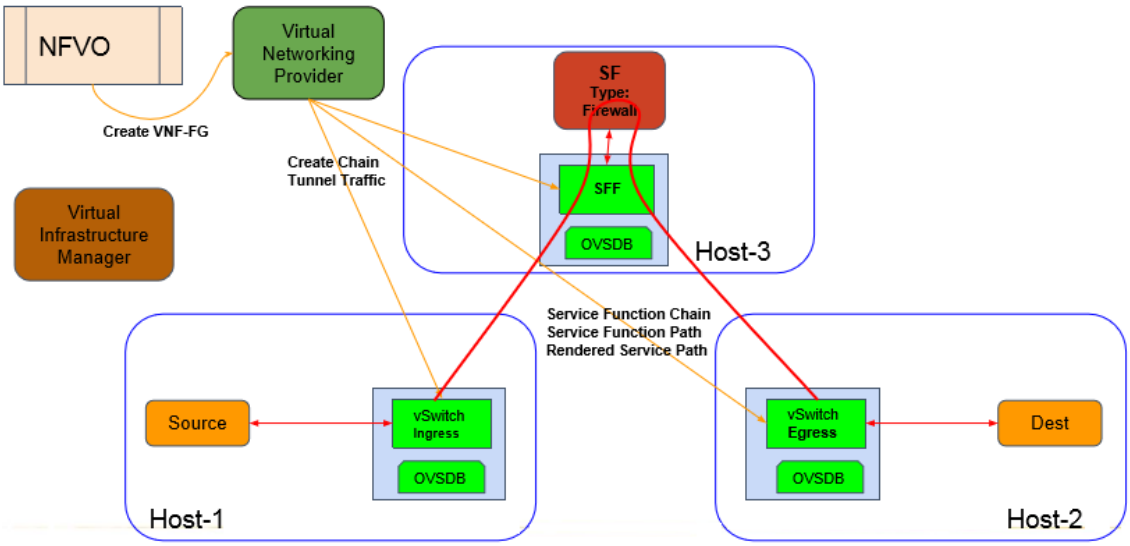

Figure 2.10. OpenvSwitch SFC role.[24]

SFC can be implemented by using the previously explained NSH or a header containing chain information can be included into other protocol. OpenvSwitch supports Network Services Header encapsulation through a patch introduced by Intel [32], although it has not been merged officially into mainline OVS. The patch supports two types of encapsulation: ETH+NSH or VxLAN-GPE+ETH+NSH.

The networking-sfc project includes an OVS agent that establishes VxLAN or GRE tunnels to transport traffic between compute nodes and uses a MPLS header to transport the chain path identifier and chain hop index.

CHAPTER 3. OPEN SOURCE MANO

Open Source MANO (OSM) is aimed to be a production-quality management and orchestration stack for NFV environments [33]. It is an open source project developed by several operators and is closely aligned with the ETSI NFV information model explained in previous chapters. It is intended to be used by VNF vendors, system integrators and operator environments.

OSM has produced three releases: release ONE on October 2016, release TWO on April 2017 and release THREE on November 2017.

Among its many features it integrates with several SDN controllers and VIMs, includes monitoring tools for services, it is suitable for greenfield and brownfield deployments and supports multi-VIM and multi-site scenarios; interaction is provided to end-users and administrators through a GUI, CLI or a Python based client library accessible via REST interfaces.

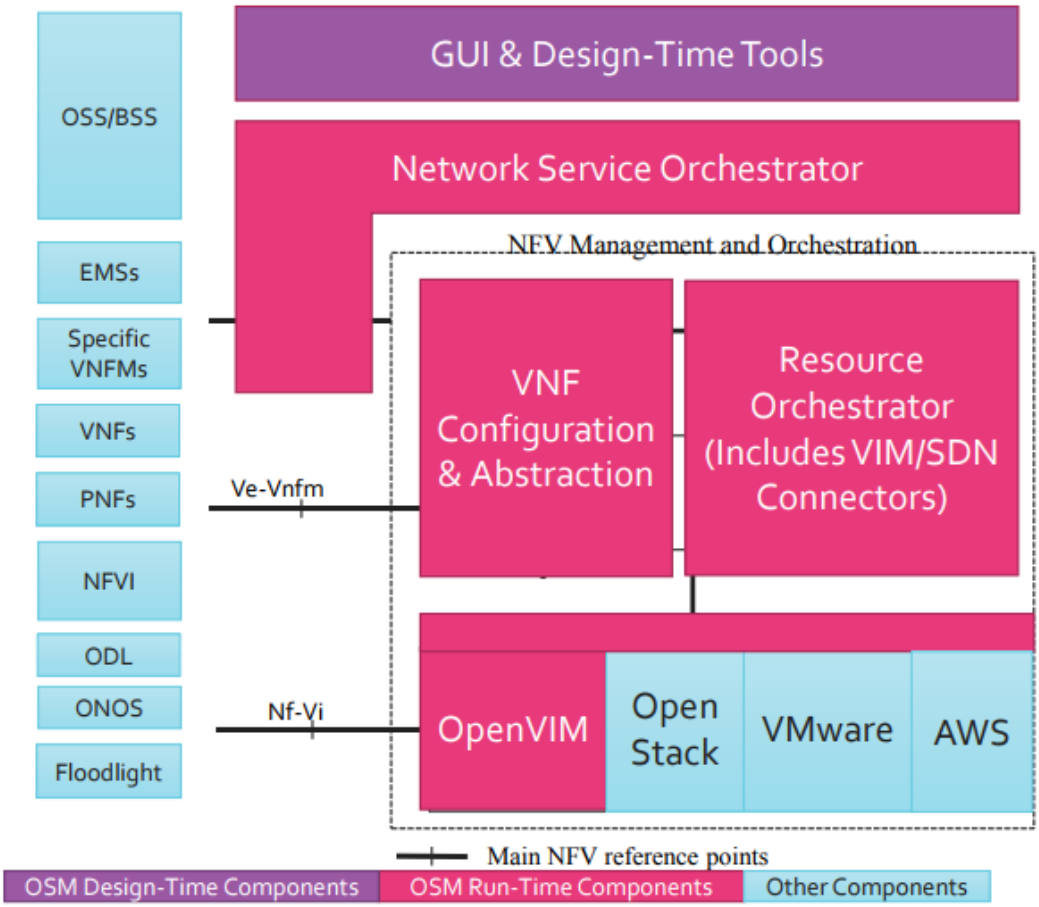

Figure 3.1. Mapping between OSM and ETSI NFV MANO. [33]

3.1. OSM Architecture

OSM logically groups its components into design-time components and run-time components; run-time components refer to operational processes involved in the management of network services while design-time is related with the data models employed, network service and virtual network function definitions, on-boarding and deployment.

The core components involved in the software stack are:

- *User Interface* (UI): This is a graphical user interface, referred as Launchpad. This component provides users with the ability to manage lifecycle operations on Virtual Network Functions and Network Services, provide statistics and graphic representation of network topologies, provides a VNF package generator, a VNF/NS catalogue composer, and other functionalities. The OSM client provides a client to remotely interact with OSM northbound REST API which incorporates a Python Functional library.
- *Service Orchestrator* (SO): this module is in charge of operational processes related to the lifecycle phases involved in deploying NFV-based network services, it governs workflow through OSM. It is comprised by the following sub-components:
  - Configuration data-store: stores the service orchestrator states, particularly VNF/NS deployment records.
  - Network Service Composition Engine: validates that VNF/NS descriptors comply with the proper YANG schema.
  - Catalogue Manager: performs create, read, update and delete operations on VNF/NS packages.
  - Plugin Interfaces: interfaces to provide interaction with the resource orchestrator and VNF configuration and abstraction layer.
- *VNF Configuration and abstraction layer* (VCA): It is responsible for enabling configurations, actions and notifications to and from the VNFs and the element managers. OSM default installation includes a generic VNFM manager, JUJU [34].
- *Resource Orchestrator* (RO): provides core services for the service orchestrator to consume, it is responsible for managing and coordinating resource allocations across the available virtual infrastructure managers and SDN controllers; it performs these operations through plug-ins that connect to the VIM/SDN controller interfaces, available plugins connect to: AWS, Vmware vCloud, Openstack, ONOS, OpenDaylight and Floodlight.
- *Monitoring*: the latest release includes this experimental module, it aims to correlate telemetry related to virtual machines and VNFs with relevant network services associated to them. It does so by leveraging on existing monitoring systems and uses Apache Kafka [35] as the message bus implementation. It integrates with Openstack Aodh, Openstack Gnocchi, Amazon CloudWatch and Vmware vRealize.

A detailed component representation follows:

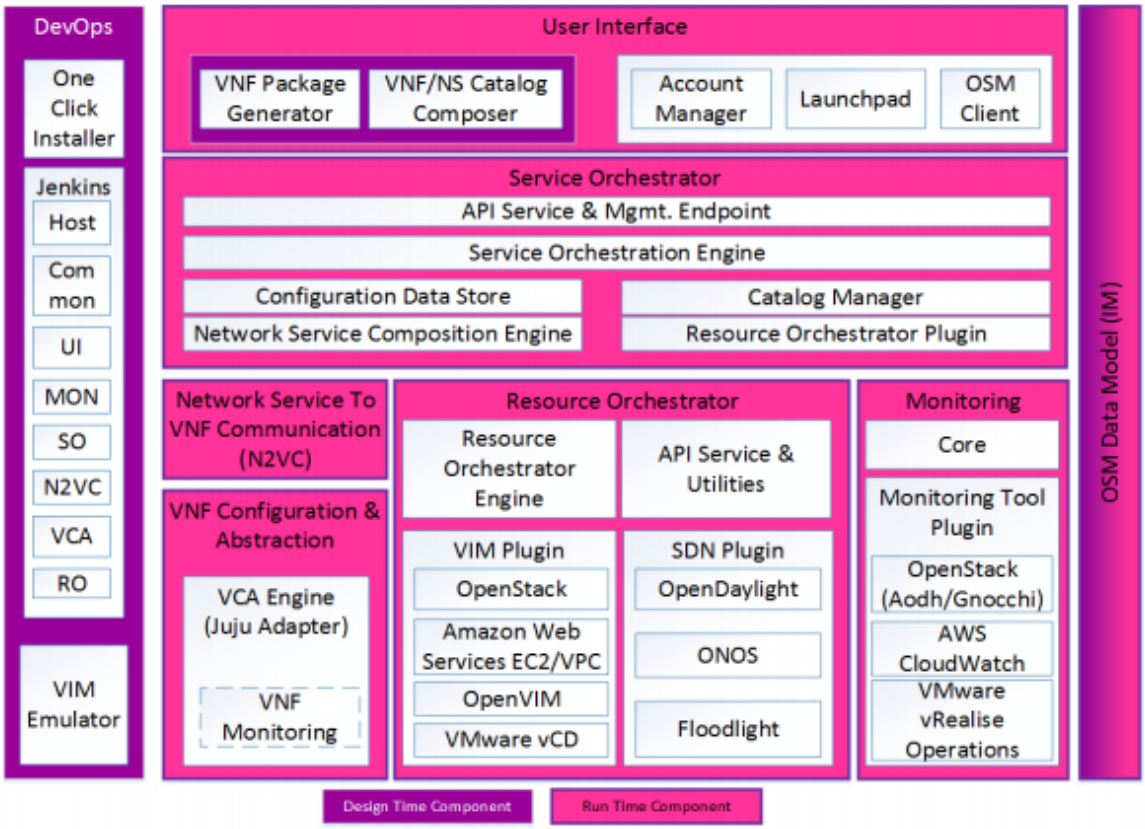

Figure 3.2. OSM Release THREE Architecture. [33]

3.2. OSM deployment

Open Source MANO is designed to be deployed as an All-in-one solution over a single server which requires having IP connectivity to the components it is to be associated as Virtual Network Functions, SDN Controllers and Virtual Infrastructure Managers. OSM will be used to deploy and manage these VNFs over the infrastructure provided by the VIMs, therefore it should have access to the overlay network that these virtual machines will use to access the external network.

OSM provides three installation options [36], it can be installed from binaries, from source or from Linux Containers (LXD) images.

OSM requires IP connectivity to the VIM and to the overlay network where VNFs will be deployed over the VIM. The same server where it is installed comes with a web server that provides the UI through https.

It can be installed from binaries, from source or from LXD images [37]. Recommended server specifications are the same for binaries and LXD images: minimum physical requirements are 4 CPUs, 8 GB RAM, 40 GB disk and recommended are 8 CPUs, 16 GB RAM, 80 GB disk. To do it from source the

recommended requirements are the same, varying only on disk amount which goes to 100 GB. The Operating System to be used is Ubuntu 16.04 configured to run LXD containers.

System deployment installs three containers that have all the components mentioned in the previous sections. These containers are called: RO (Resource Orchestrator), VCA (VNF configuration and abstraction layer), and SO-ub (Service Orchestrator and user interface). Although it is in the roadmap that the user interface services are to be separated from the service orchestrator. These containers communicate between them and to the external network through linux bridge.

In order to interact with VIMs these must be configured for external http API access and added to OSM information database, the minimal information to be provided is the VIM name, authentication credentials, authentication URL, tenant name and account type. The following ports should be opened in the server containing OSM for correct operation: 8443, 8000, 4567, 8008, 80 and 9090.

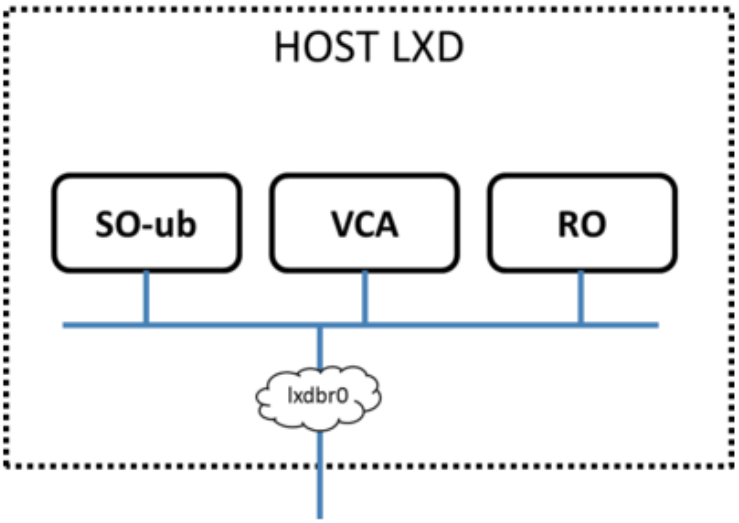

Figure 3.3. OSM logical software distribution. [36]

CHAPTER 4. THEORETICAL RESOURCE ASSIGNMENT OPTIMIZATION

In SDN/NFV environments the presence of an orchestrator poses an advantage since it will provide an overview of network services status and operation, thus giving network administrators the ability to place their workloads efficiently. In order to achieve this, the orchestrator needs to have a complete view of the services deployed and the available infrastructure resources for future demands.

Therefore, service visibility and telemetry metrics are paramount to achieve the initial goals proposed by the NFV architecture in relation to network operation and management.

The orchestrators studied in previous chapters, Open Source MANO and Tacker, are still a work in progress in this area, since they provide no superior intelligence in resource assignment compared to the one already provided by the Virtual Infrastructure Manager. In this case study specifically, the VIM analyzed was Openstack.

This chapter introduces a resource assignment optimization framework that will enable an orchestrator to leverage certain algorithms to efficiently place virtual machines into the cloud infrastructure according to real-time metrics describing the infrastructure status. In order to do this, the orchestrator needs to have telemetry measures that reliably describe in real time the resource status and specific implementations of the algorithms based on the resources to be measured. The aforementioned metrics need to be relevant in network and cloud computing environments, as is the memory usage, CPU usage and network bandwidth consumption. The algorithms to be studied for this case will be bin-packing and stochastic knapsack.

4.1. Openstack Resource Assignment

In these orchestrators resource instantiation is done by using directly the APIs provided by Openstack's compute project Nova, which sub sequentially handles the placements of the virtual machines according to internal scheduling provided as a Nova service. The Nova-scheduler component determines on which physical host a Virtual Machine is to be hosted.

Nova scheduler's default configuration is to first filter the available hosts according to the requested resources and after to select where to deploy the virtual machine. In order to filter instances, it verifies that the hosts meet the following requirements: availability zone, RAM, disk, CPU and image properties; this produces a list of capable hosts to place the loads.

Afterwards every host in the list is weighted using a cost defined as a relation between the host capabilities and the requested resources by the instance. Hosts with the highest weight will be given the higher priority and this priority defines which host will take the requested load. The default behavior of this algorithm is to assign weights such that the scheduler spreads instances across all hosts evenly. The weighting parameters can be modified through Nova service configuration.

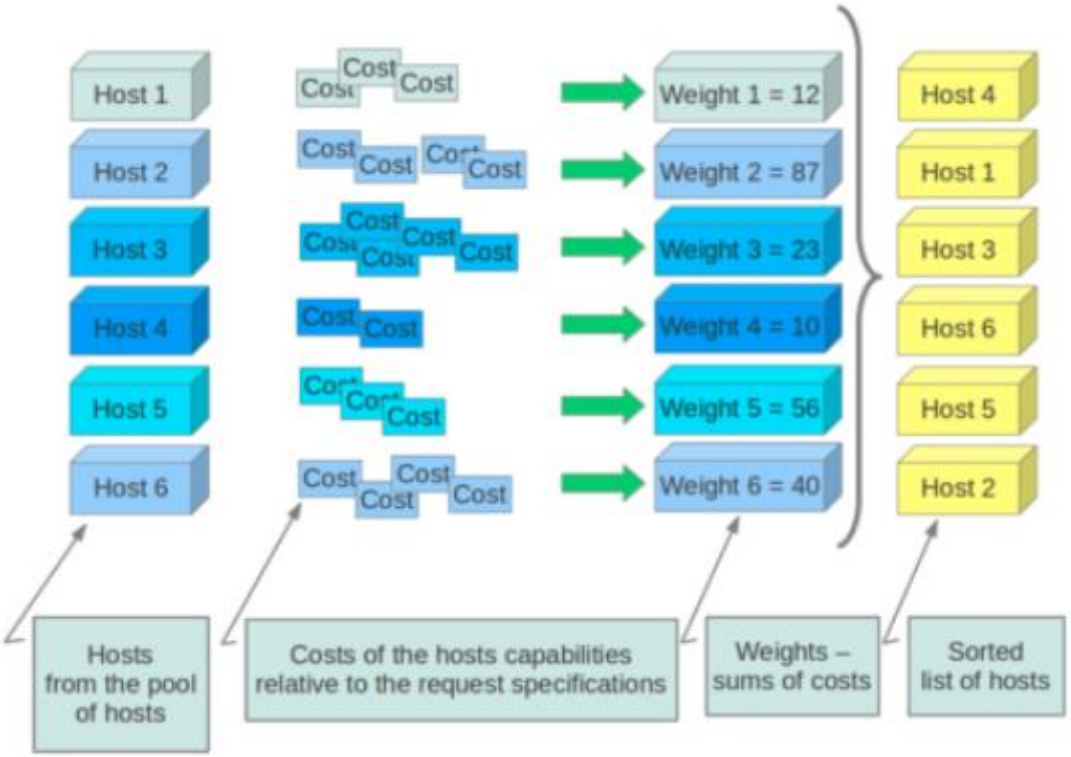

Figure 4.1. Nova Host Weighting. [38]

4.2. Optimization Algorithms

Optimization algorithms are methods designed to find the closest optimal solution available for a problem, these problems can be divided into two categories whether the variables are continuous or discrete. Problems using the latter type of variables are called combinatorial problems.

In NFV and cloud environments resources such as memory, disk capacity, CPU and network capacity are discrete variables; the problem's objective is to find the best way to allocate this resources in the physical infrastructure available, therefore we are handling discrete values. Another aspect to consider is how the requests will arrive to the system administrators; there are two scenarios to be considered:

- Instance requests arrive one at a time, this kind of scenarios are known as on-line algorithms.

- All requests are known beforehand, which is referred as off-line algorithms.

To measure the effectiveness of an optimization algorithm the measure time complexity is used, which refers to the number of elementary operations that the algorithm undergoes. The time taken by the algorithm to complete can be estimated by knowing how long does an elementary operation takes; since this time depends on the nature of the input the worst-case scenario is the usual consideration. Time complexity is expressed using big O notation.

4.2.1. Bin-packing Algorithms

Bin packing problems consist of objects of different volumes that must be packed into a finite number of bins or containers each of a specific volume such that the number of bins used is minimal. Therefore, the nature of the problem varies according to the nature of the objects and bins to be used, examples of study cases would be two-dimensional packing, linear packing, packing by weight, among others.
These problems have several real life applications like loading containers, job scheduling, placing data on disks or assigning groups into vehicles. In our case we would be assigning virtual machines into physical hosts.

4.2.2. One-dimension Packing

One-dimension packing problem are situations where there are “n” items of different weights and bins each of capacity “c”, the objective is to assign each item to a bin such that the number of total used bins is minimized [39]. The lower bound of the solution is the following:

$$Min \leq ceil(TotalWeight/BinCapacity)$$

In this case consider following Online Algorithms:

- *Next fit*: It selects an initial bin and tries to fit each item into it, if the next item does not fit in the current evaluated bin it uses a new bin. After an item does not fit into this bin it is not taken into consideration for further items. Time complexity for this algorithm is $O(n)$
- *First fit*: It selects an initial bin and tries to fit each item into it, if the next item does not fit in the current evaluated bin it will use a new bin. For each item every available bin is considered as a candidate as storage, so there is an additional iteration when compared with next fit. Time complexity for this algorithm is $O(n^2)$
- *Best fit*: It selects an initial bin and tries to fit each item into it, to decide where to put the item it will take into account the empty space left on every bin and will put the item in the one that leaves the smallest empty space left. In the case that no bin can store the object it will create a new bin. Time complexity for this algorithm is $O(nlog(n))$

In the previous cases non-optimal solutions will be found if large items occur late in the sequence. A better approach can be done when all items are known, in such case there is the following off-line Algorithm:

- First fit decreasing (FFD): The input sequence of objects is sorted by placing the larger items first. After this is done the first fit algorithm is executed. Time complexity for this algorithm is $O(nlog(n))$
- Best fit decreasing (BFD): The input sequence of objects is sorted by placing the larger items first. Afterwards the best fit algorithm is executed. Time complexity for this algorithm is $O(nlog(n))$

4.2.3. Two-dimensional Packing

In two-dimensional bin packing problems bins and objects can be considered to be rectangular, therefore having a certain width and height. The problem's objective is to pack all the items into the minimum number of containers. The algorithms used for these situations can be divided into two categories: one phase algorithms that directly put items into the bins and two phase algorithms that start by packing items into a single strip that has infinite height, afterwards using this solution for packing the items into finite bins. All algorithms to be considered are off-line.

The first method used in two phase algorithms is called strip packing, it considers a strip of finite width but infinite height and a set of rectangular items with width smaller than the container's. The objective is to pack all the items minimizing the height used, a common approach is level-oriented where items are packed from left to right forming vertical levels that are defined by the height of the tallest item on the previous level [40]. The following algorithms sort items by decreasing height and are level-oriented:

- *First-fit Decreasing Height* (FFDH): Selects an initial level and tries to pack the next item on this level, being its width the delimiting factor. If this level cannot accommodate the item a new level is created. For every item all available levels are considered, inserting it on the first level that can hold it. Time complexity for this algorithm is $O(nlog(n))$
- *Next-fit Decreasing Height* (NFDH): Selects an initial level and tries to pack the next item on the current level being evaluated, being its width the delimiting factor. If the current level cannot accommodate the item a new level is created. Once a level is created others are not taken into consideration for future items. Time complexity for this algorithm is $O(nlog(n))$
- *Best-fit Decreasing Height* (BFDH): Selects an initial level and tries to pack the next item on this level, being its width the delimiting factor. If this level cannot accommodate the item a new level is created. For every item all available levels are considered, inserting it on the one where it will leave the least amount of space available.

The following figure describes the behavior of each of these algorithms for a bin with width equal to 10, and a request composed by: Requests width (reference number, width): {(1,4), (2,2), (3,6.5), (4,2), (5,3), (6,1)}.

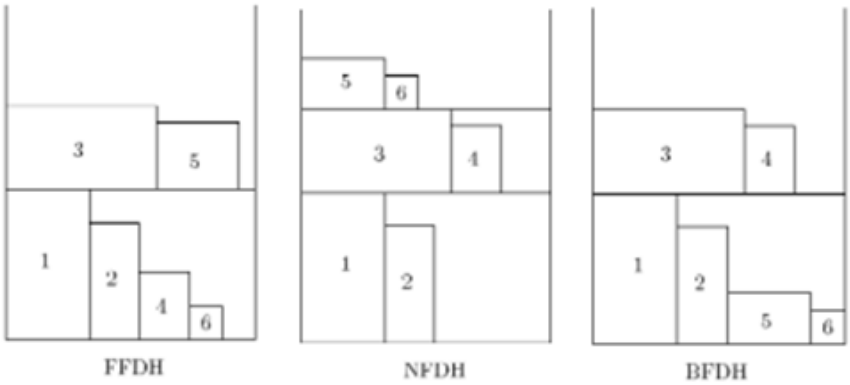

Figure 4.2. Strip Packing Algorithms examples. [41]

Two-phase bin-packing algorithms make use of the previous methods, first obtaining several levels from strip packing and afterwards obtaining a finite solution by solving a one-dimensional bin-packing solution with a fixed height. The algorithms considered follow:

- *Hybrid First-Fit* (HFF)[42]: In the first phase a strip packing is obtained by performing the FFDH algorithm. So we will have several strips with different heights. On the second phase a First-fit decreasing algorithm is performed, this sorts the strips obtained beforehand in decreasing height. Time complexity for this algorithm is $O(n.log(n))$

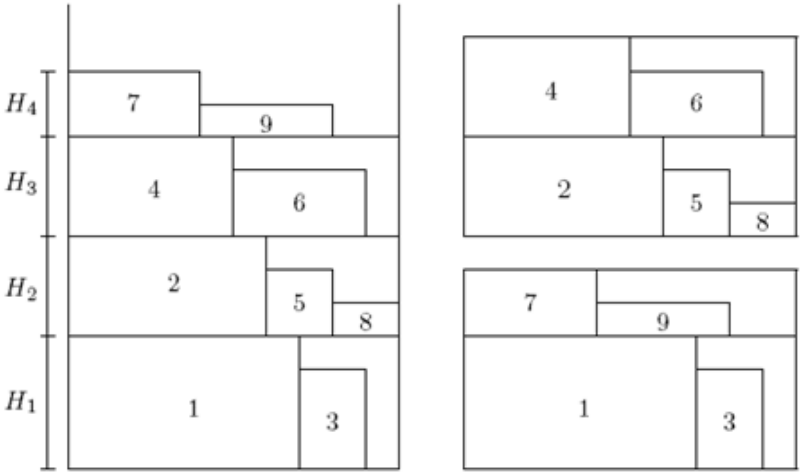

Figure 4.3. Hybrid First-Fit example. [42]

- Hybrid Next-Fit (HNF)[43]: In the first phase a strip packing is obtained by performing the NFDH algorithm. On the second phase a Next-fit decreasing algorithm is performed this sorts the strips obtained

beforehand in decreasing height. Time complexity for this algorithm is $O(nlog(n))$

- Hybrid Best-Fit (HBF)[44]: In the first phase a strip packing is obtained by performing the BFDH algorithm. On the second phase a Best-fit decreasing algorithm is performed this sorts the strips obtained beforehand in decreasing height.

One phase two-dimension packing algorithms are:

- *Finite First-Fit* (FFF) [44]: This approach adopts a First-Fit decreasing algorithm (FFDH) where an item is packed on the lowest vertical level of the first container where it fits, if no level can accommodate it a new vertical level is created in the first bin if the height is still sufficient otherwise a new level is created in a new bin. For every item all available levels are considered, inserting it on the first level that can hold it. Time complexity for this algorithm is $O(nlog(n))$.

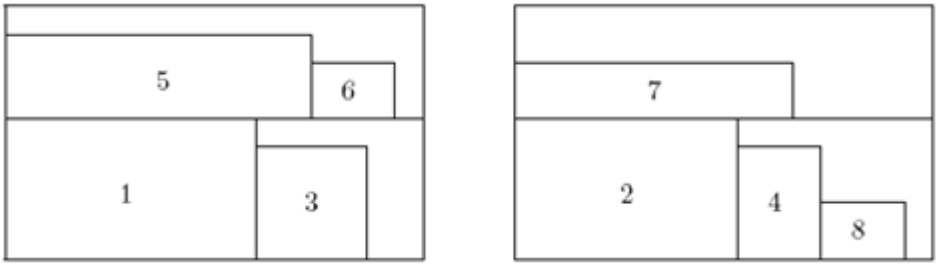

Figure 4.4. Finite First-Fit example. [44]

- *Finite Next-Fit* (FNF)[44]: This packs items using a Next-Fit Decreasing Algorithm (NFDH) packing an item in the lowest vertical level of the first container where it fits, if no level can accommodate it a new vertical level is created in the first bin if the height is sufficient, otherwise a new level is created in a new bin. Once a level is created others are not taken into consideration for future items. Time complexity for this algorithm is$O(n.log(n))$.
- *Finite Bottom-Left* (FBL)[44]: This method does not use vertical levels, this algorithm initially sorts items by decreasing width. The evaluated item is packed in the lowest position of any existing bin, justified to the left; if there is no space on any bin a new one is started.
- *Next Bottom-Left* (NBL)[44]: This algorithm works as the described previously, but at any time only one bin is available for packing. Once an item does not fit into this bin it is not considered for future items anymore.

CHAPTER 5. PRACTICAL RESOURCE ASSIGNMENT OPTIMIZATION

In order to build a resource assignment framework for the previously described components of and SDN/NFV environments, we approach it as two different problems: the telemetry data collection problem and an optimization problem.

The first part of the problem aims at collecting data related to the status of the physical resources that are a part of the cloud and storing them locally so the orchestrator can access this information as quickly as possible whenever needed.

The second part will implement resource assignment intelligence on the orchestrator to allocate virtual physical and virtual resources making the most efficient use of the available infrastructure. For this it will access the information containing resource availability and take decisions about resource placement.

The virtual infrastructure manager used is Openstack and the technologies leveraged for the development of these components are the same ones as employed in Open Source MANO: python programming language and MySQL database.

5.1. Openstack Resource Collection

A python script was coded which leveraged on Openstack APIs to gather telemetry data and sequentially saving it on a MySQL data base. Therefore, the telemetry service referred as Ceilometer has to be set up in an operating Openstack implementation in order to collect resource status data. The logic operation of the script is the following:

i. Get Openstack API credentials from environment variables.
ii. Establish a connection to Openstack Controller.
iii. Get the client for the API to be consumed, in this case Ceilometer client.
iv. Connect to the database.
v. Declare a matrix variable that is to be the data structure used to collect telemetry data.
vi. Get each sample and extract the information to be stored on the database: resource ID, Volume, recorded time, sample name, unit of measure and project ID.
vii. Insert information in database or update existing records.

The created database holds all the relevant information for each sample obtained, it is designed to hold the last 10 values obtained for each measurement. The structure is the following:

Table name	openstack_resources
Column 1 (primary key)	resource_id varchar (120)
Column 2	name varchar (40)
Column 3	Value1 varchar (20)
Column 4	value2 varchar (20)
Column 5	value3 varchar (20)
Column 6	value4 varchar (20)
Column 7	value5 varchar (20)
Column 8	value6 varchar (20)
Column 9	value7 varchar (20)
Column 10	value8 varchar (20)
Column 11	value9 varchar (20)
Column 12	value10 varchar (20)
Column 13	unit varchar (20)
Column 14	Time varchar (30)
Column 15	instance_id varchar (60)
Column 16	user_id varchar (60)
Column 17	host_id varchar (60)
Column 18	max_value varchar (20)
Column 19	min_value varchar (20)
Column 20	project_id varchar (60)
Column 21	Average varchar (20)

Table 5.1. Database structure.

A brief explanation for each of the relevant values follows:

- Resource_id: alphanumerical string assigned by Openstack to identify its resources.
- Name: Human readable reference name for the measurement.
- Value: Most recent 10 measurements taken for each resource.
- Unit: unit of measurement.
- Time: timestamp related to measured value.
- Instance_id: ID of the instance to which the measurement is associated.
- User_id: ID of the user to which the measurement is associated.
- Host_id: ID of the physical host to which the measurement is associated.
- Max_value: maximum value recorded for a certain measurement.
- Min_value: minimum value recorded for a certain measurement.
- Project_id: ID of the project to which the measurement is associated.
- Average: Average of the last 10 measurements taken.

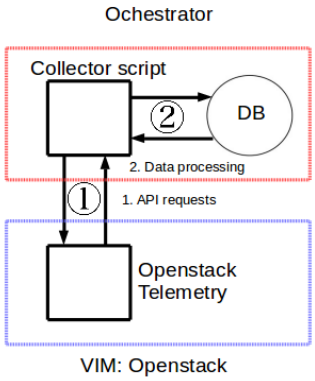

Figure 5.1. Openstack Resource Collection Components.

5.2. Openstack Resource Assignment

In order to model the resource assignment problem that could potentially arise in a production SDN/NFV environment a python-based script implementing a set of resource assignment algorithms was developed. The aforementioned script works under the premise that the resource requests made to the orchestrator are already known, therefore they can be manipulated before actually allocating resources; this would put us under an off-line bin-packing scenario.

The algorithms coded and tested are: First-Fit decreasing Height, Best-Fit decreasing Height, Hybrid First-Fit and Hybrid Best-Fit. As a starting point the approach was a two-dimensional problem, based in physical CPU and memory resources. An initial assessment of the problem can be graphically described in three steps:

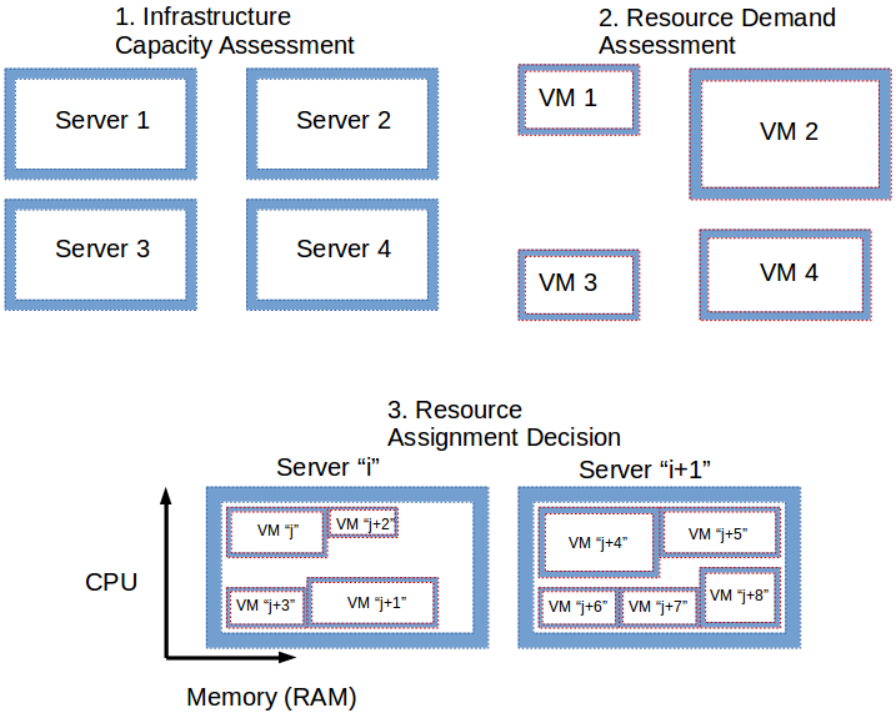

Figure 5.2. Initial Resource Assignment Assessment.

The workflow followed by one-phase algorithms implemented (FFDH, BFDH) is the following:

i. Sort the resource requests on a particular feature, either CPU or memory.

ii. Loop through each virtual machine requested for assignment.

    i. Extract resource requirement for each virtual machine.

    ii. Loop through each server

        i. Compare requested resources with resources available on the server being evaluated

        ii. Apply first-fit or best-fit logic and assign VM to selected server. In order for a VM to be allocated it must be able to fit in the server on both parameters CPU and memory

The workflow applied to two-phase algorithms used (HFF, HBF) is as follows:

i. Sort the resource request on a particular feature, either CPU or memory.

ii. Divide server capacity by half, obtaining the double amount of servers in relation to what actually is available.

iii. Perform FFDH or BFDH on the request and the new server set.

iv. Sort servers based on the amount of resources assigned in a decreasing manner.

v. Pair servers with most resources assigned with the ones with least resource assignment.

vi. Loop through unassigned virtual machines trying to fit them on any server that has enough resource availability.

5.3. Results

To get a performance evaluation of how these algorithms behaved under various circumstances a series of tests were performed on the scripts explained on the previous section.

As resource availability the assumption is that there is a three node compute Openstack cloud, with each compute node providing 12 CPUs and 64 GB of RAM memory. Thus the cloud provides 36 CPUs and 192 GB of RAM.

In order to make a performance assessment of resource allocation the following scenarios where simulated; only varying the resource requests. Virtual Machine

flavors where taken based on common flavors seen in cloud computing scenarios:

- A request made of identical virtual machines having 1 CPU and 2 GB RAM, increasing the amount of virtual machines requested on each test.
- Make a request that encompasses all the available CPUs but does not stress the amount of memory.
- Make a request that fills up all available memory but does not stress the amount of CPUs.
- Make a request that is equivalent to all the available CPUs and memory.
- For each of the previous cases increment by 10% the amount of resources requested.
- Finally, a randomly generated request sequence was generated, this was produced by having a set of possible virtual machine flavors with each one corresponding to a certain number. At the beginning of the script a certain amount of random numbers is generated, each one mapping to a flavor of virtual machine. There were two types of scenarios simulated: generate random requests that accumulate for a total capacity similar to two servers and accumulating for a total capacity similar to all available servers.

After evaluating the results, the first case was found to be non-relevant since identical requests are assigned uniformly to servers, filling them up one by one until one resource is exhausted. Therefore, the amount of servers able to fit will just depend on the dominating resource on the request in comparison to the available resources.

The nomenclature used for each algorithm is as follows, it is comprised by the algorithms name followed by the parameter by which it is sorted; this parameter can be either CPU or memory. Therefore, the algorithm Best-Fit Descending Height sorted by CPU will be referred as BFDH CPU and sorted by memory will be BFDH MEM.

The following tables will describe the results obtained from the first set of scenarios. These terms are used in the aforementioned tables to reference the executed use cases:

- Each row represents a VM composed by a number of CPUs and RAM memory, these are referenced as “CPU” and “MEM” (expressed in KB).
- CPU loaded are requests that will fill CPU in greater measure than memory, memory loaded is the inverse and balanced will fill it similarly.
- For each type of request there are two executions: regular executions maintain the values indicated in the table, and overloaded increase these values by 10% in order to see how the algorithm handles receiving a request that includes more resources that there is available.
- Unassigned row depicts how many VMs are unassigned for each case because server capacity is not enough.

Request 1 – CPU loaded	
CPU	MEM
1	512
1	512
1	512
1	1024
1	1024
1	1024
3	16384
2	12288
1	4096
1	512
1	512
1	512
1	1024
1	1024
1	1024
3	16384
2	12288
1	4096
1	512
1	512
1	512
1	1024
1	1024
1	1024
3	16384
2	12288
1	4096

Total	
36	112128

Request 2 – Memory Loaded	
CPU	MEM
1	12288
1	12288
1	16384
1	4096
1	8192
1	4096
1	8192
1	12288
1	12288
1	16384
1	4096
1	8192
1	4096
1	8192
1	12288
1	12288
1	16384
1	4096
1	8192
1	4096
1	8192

Total	
21	196608

Request 3 – Balanced	
CPU	MEM
1	4096
2	6144
1	2048
2	12288
2	16384
1	4096
2	8192
1	12288
1	4096
2	6144
1	2048
2	12288
2	16384
1	4096
2	8192
1	12288
1	4096
2	6144
1	2048
2	12288
2	16384
1	4096
2	8192
1	12288

Total	
36	196608

Table 5.2. Regular resource assignment scenarios simulated.

Regular	
Server Distrib – FFDH CPU	
12	61952
12	33792
12	16384
Unassigned	

Server Distrib – FFDH MEM	
12	65536
12	38912
12	7680
Unassigned	

Server Distrib – BFDH CPU	
12	61952
12	33792
12	16384
Unassigned	

Server Distrib – BFDH MEM	
12	65536
12	38912
12	7680
Unassigned	

Regular	
Server Distrib – FFDH CPU	
7	65536
7	65536
7	65536
Unassigned	

Server Distrib – FFDH MEM	
5	65536
6	65536
10	65536
Unassigned	

Server Distrib – BFDH CPU	
7	65536
7	65536
7	65536
Unassigned	

Server Distrib – BFDH MEM	
5	65536
6	65536
10	65536
Unassigned	

Regular	
Server Distrib – FFDH CPU	
12	61440
12	65536
11	57344
Unassigned	1

Server Distrib – FFDH MEM	
9	65536
8	65536
12	43008
Unassigned	7

Server Distrib – BFDH CPU	
12	61440
12	65536
11	57344
Unassigned	1

Server Distrib – BFDH MEM	
9	65536
8	65536
12	43008
Unassigned	7

Table 5.3. Regular requests results for one-phase algorithms.

Server Distrib – HFF CPU	
12	37376
12	33792
12	40960
Unassigned	

Server Distrib – HFF CPU	
7	65536
7	65536
7	65536
Unassigned	

Server Distrib – HFF CPU	
12	61440
11	65536
12	57344
Unassigned	1

Server Distrib – HFF MEM	
12	37376
12	65536
12	9216
Unassigned	

Server Distrib – HFF MEM	
5	65536
9	65536
7	65536
Unassigned	

Server Distrib – HFF MEM	
9	65536
12	57344
12	65536
Unassigned	3

Server Distrib – HBF CPU	
12	37376
12	33792
12	40960
Unassigned	

Server Distrib – HBF CPU	
7	65536
7	65536
7	65536
Unassigned	

Server Distrib – HBF CPU	
12	61440
11	65536
12	57344
Unassigned	1

Server Distrib – HBF MEM	
12	37376
12	65536
12	9216
Unassigned	

Server Distrib – HBF MEM	
5	65536
9	65536
7	65536
Unassigned	

Server Distrib – HBF MEM	
9	65536
12	57344
12	65536
Unassigned	3

Table 5.4. Regular requests results for two-phase algorithms.

10% more	
Server Distrib – FFDH CPU	
12	61952
12	33792
12	16384
Unassigned	3

10% more	
Server Distrib – FFDH CPU	
7	65536
7	65536
7	65536
Unassigned	3

10% more	
Server Distrib – FFDH CPU	
12	61440
12	65536
12	63488
Unassigned	3

Server Distrib – FFDH MEM	
12	65536
12	54272
12	9216
Unassigned	3

Server Distrib – FFDH MEM	
5	65536
6	65536
9	65536
Unassigned	4

Server Distrib – FFDH MEM	
9	65536
8	65536
12	43008
Unassigned	10

Server Distrib – BFDH CPU	
12	61952
12	33792
12	16384
Unassigned	3

Server Distrib – BFDH CPU	
7	65536
7	65536
7	65536
Unassigned	3

Server Distrib – BFDH CPU	
12	61440
12	65536
12	63488
Unassigned	3

Server Distrib – BFDH MEM	
12	65536
12	54272
12	9216
Unassigned	3

Server Distrib – BFDH MEM	
5	65536
6	65536
9	65536
Unassigned	4

Server Distrib – BFDH MEM	
9	65536
8	65536
12	43008
Unassigned	10

Table 5.5. Overloaded requests results for one-phase algorithms.

Server Distrib – HFF CPU	
12	37376
12	33792
12	40960
Unassigned	3

Server Distrib – HFF CPU	
7	65536
7	65536
7	65536
Unassigned	3

Server Distrib – HFF CPU	
12	61440
11	65536
12	57344
Unassigned	4

Server Distrib – HFF MEM	
12	64512
12	55296
12	9216
Unassigned	3

Server Distrib – HFF MEM	
5	65536
8	65536
7	65536
Unassigned	4

Server Distrib – HFF MEM	
9	65536
12	61440
11	65536
Unassigned	7

Server Distrib – HBF CPU	
12	37376
12	33792
12	40960
Unassigned	3

Server Distrib – HBF CPU	
7	65536
7	65536
7	65536
Unassigned	3

Server Distrib – HBF CPU	
12	61440
11	65536
12	57344
Unassigned	4

Server Distrib – HBF MEM	
12	64512
12	55296
12	9216
Unassigned	3

Server Distrib – HBF MEM	
5	65536
8	65536
7	65536
Unassigned	4

Server Distrib – HBF MEM	
9	65536
12	61440
11	65536
Unassigned	7

Table 5.6. Overloaded requests results for two-phase algorithms.

A graphical representation of these results is described on the following figures:

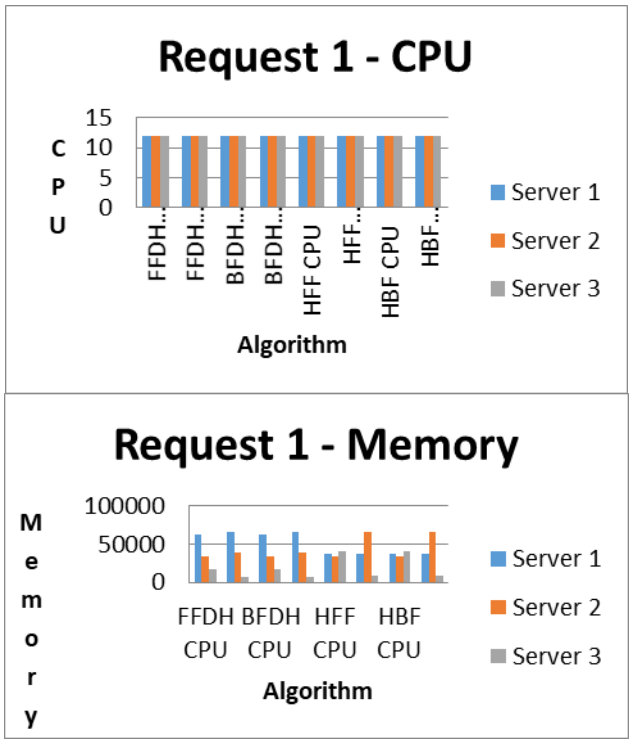

Figure 5.3. Results for first set of regular requests.

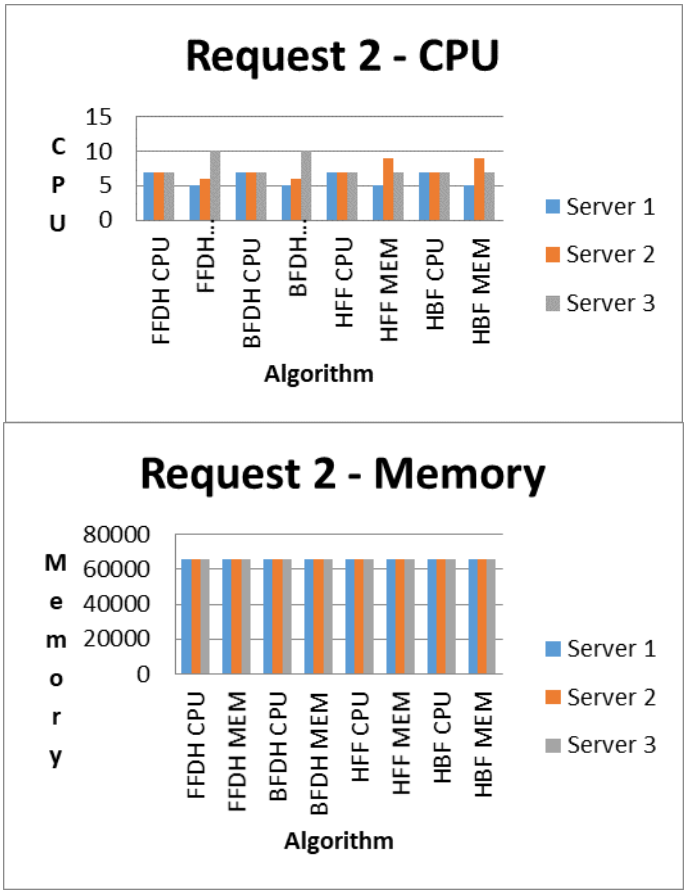

Figure 5.4. Results for second set of regular requests.

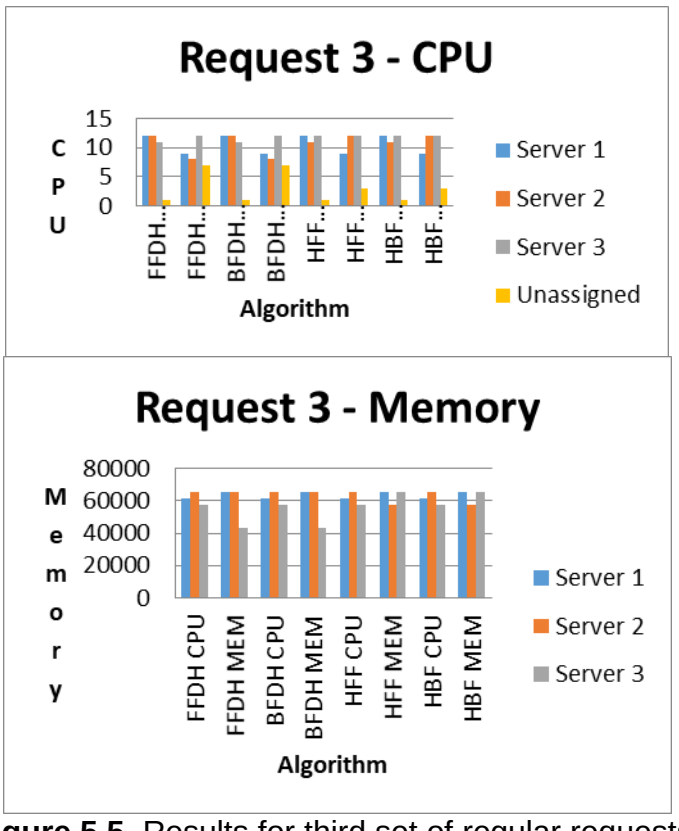

Figure 5.5. Results for third set of regular requests.

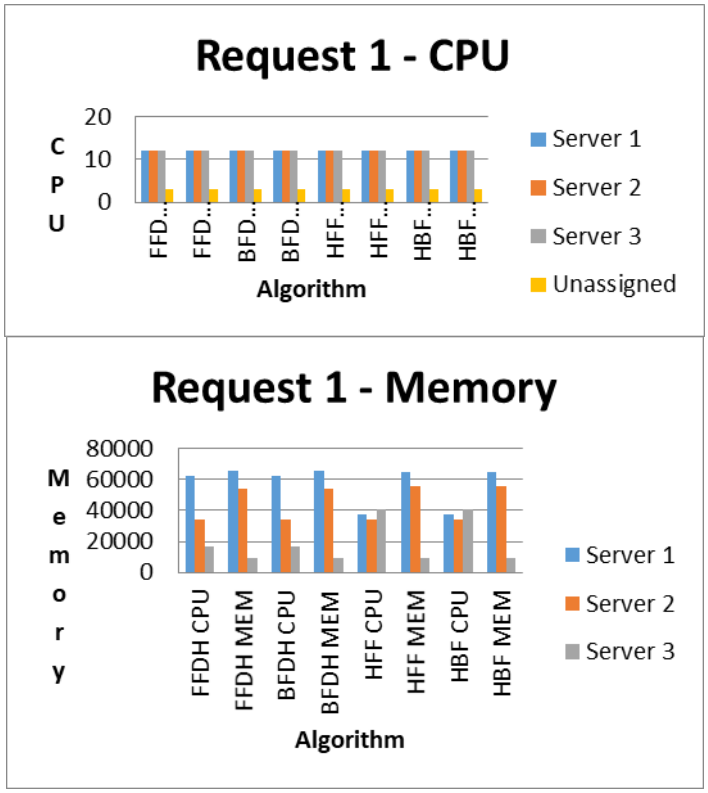

Figure 5.6. Results for first set of overloaded requests.

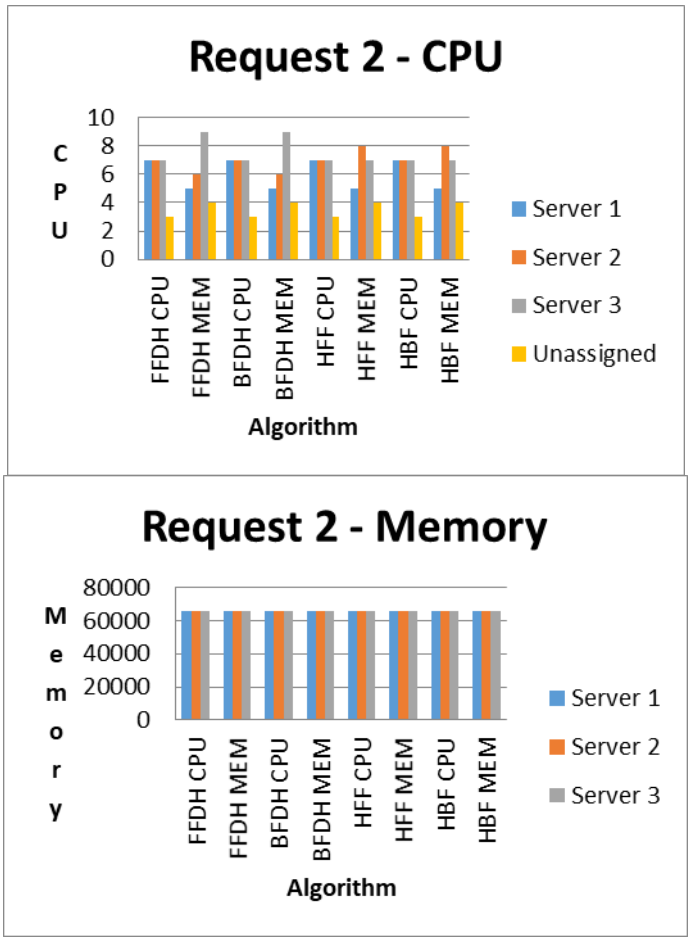

Figure 5.7. Results for second set of overloaded requests.

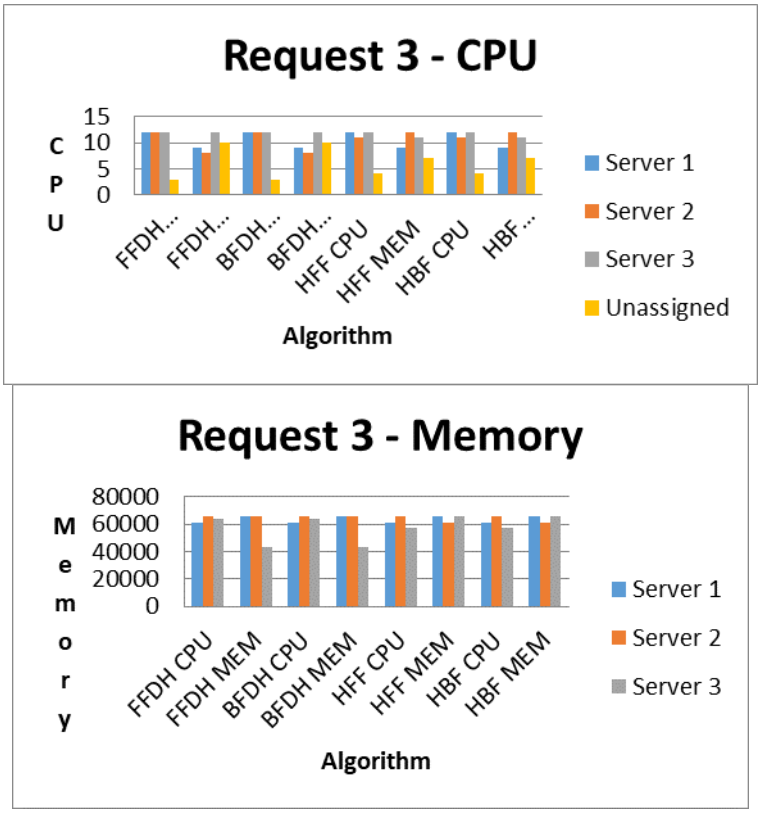

Figure 5.8. Results for third set of overloaded requests.

The requests and results for the randomly generated requests set of tests are described in the following tables:

Request 4 – Random	
3	16384
2	12288
5	25600
2	12288
2	12288
4	20480
1	6144
2	12288
2	12288
4	32768

Total	
27	162816

Request 5 – Random	
1	4096
4	20480
1	4096
5	25600
1	6144
1	6144
1	4096
3	18432
4	20480
3	18432

Total	
24	128000

Request 6 – Random	
4	32768
1	6144
1	4096
2	16384
2	12288
3	18432
1	4096
1	4096
3	16384
3	18432
3	18432

Total	
24	151552

Request 7 – Random	
1	6144
3	16384
1	6144
3	18432
5	25600
3	16384
1	4096
1	4096
2	16384
4	20480
3	16384
4	32768
4	20480
Total	
35	203776

Request 8 – Random	
2	12288
1	4096
1	4096
1	4096
2	16384
1	4096
5	25600
5	25600
4	20480
1	6144
3	18432
4	32768
2	12288
3	16384
Total	
35	202752

Request 9 – Random	
5	25600
1	6144
5	25600
2	12288
3	18432
3	16384
2	16384
1	6144
2	16384
3	18432
2	16384
3	16384
Total	
32	194560

Table 5.7. Random generated requests.

The following graphs will describe the results obtained from these set of requests:

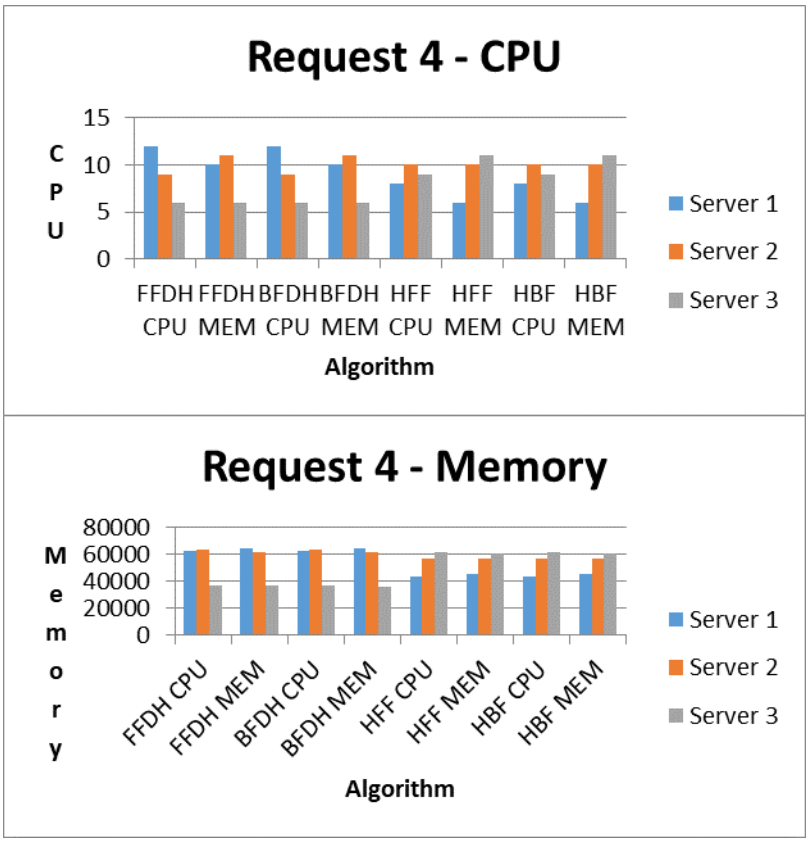

Figure 5.9. Results for fourth set of requests.

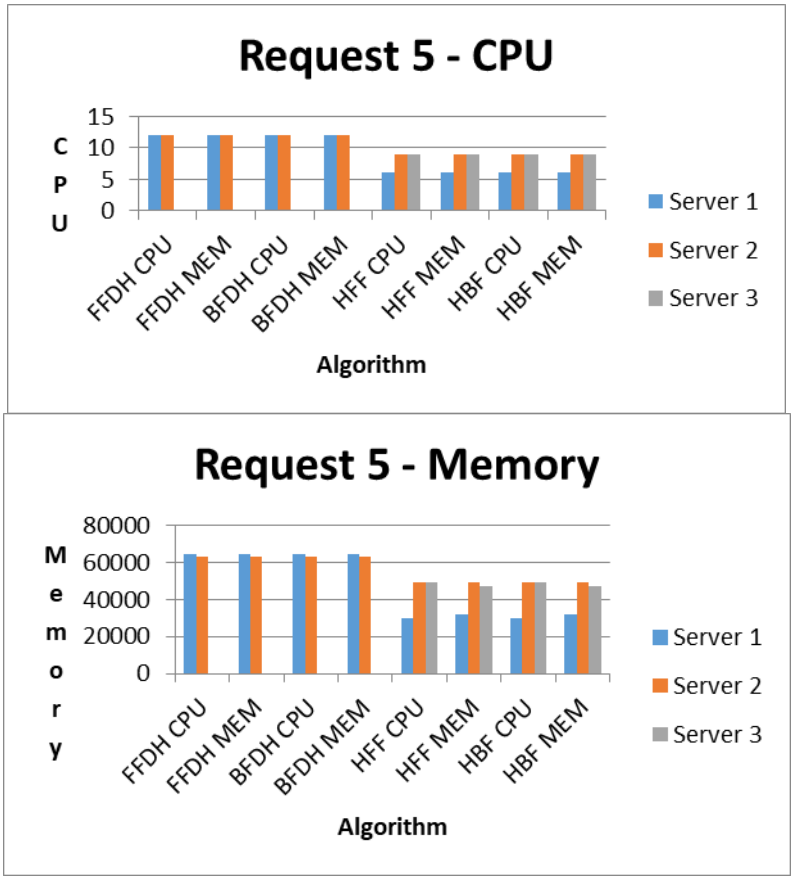

Figure 5.10. Results for fifth set of requests.

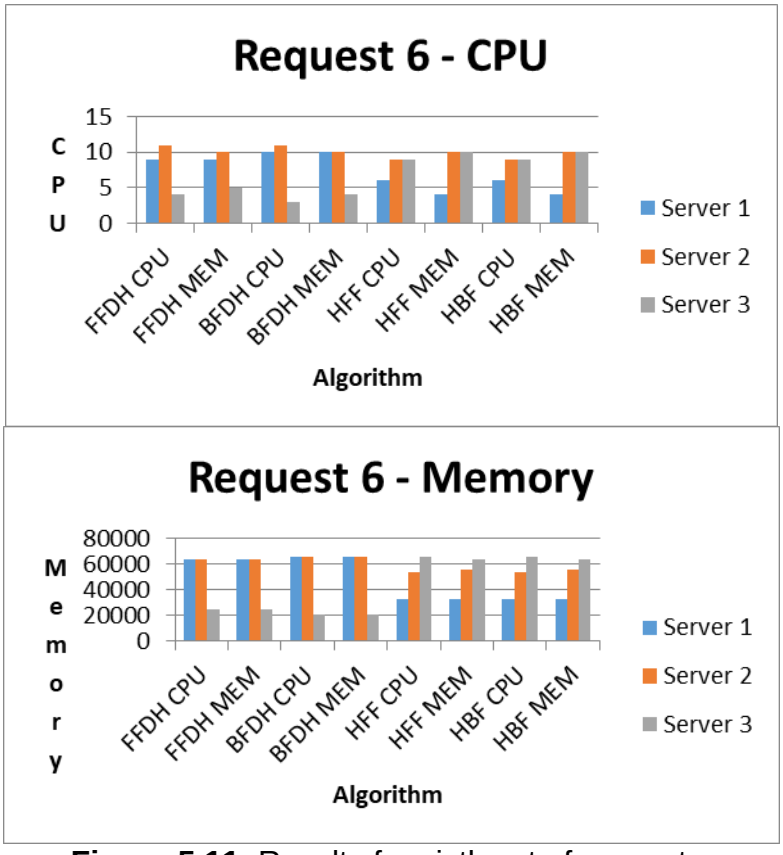

Figure 5.11. Results for sixth set of requests.

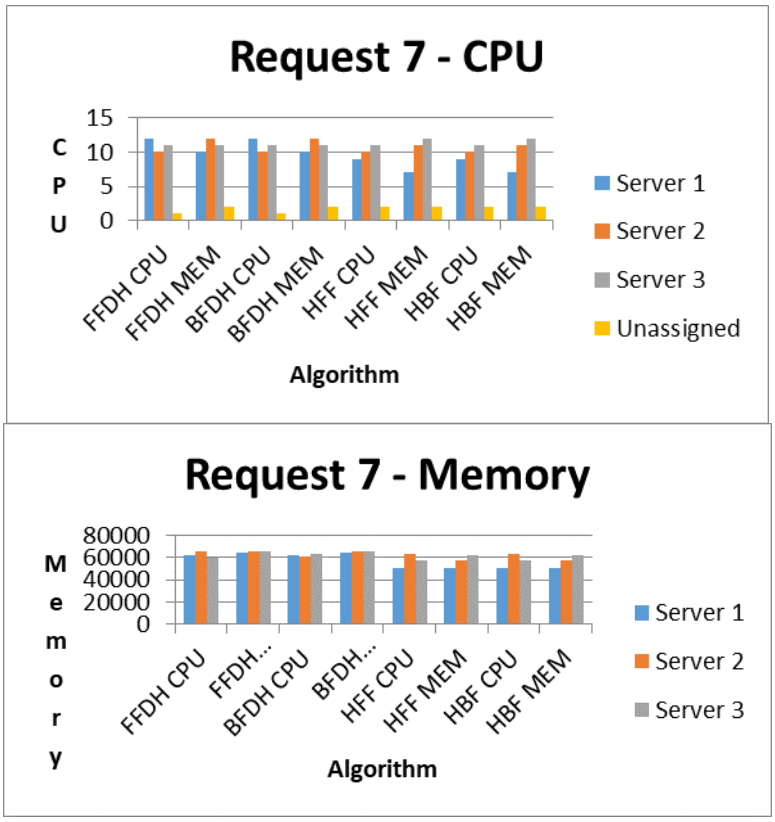

Figure 5.12. Results for seventh set of requests.

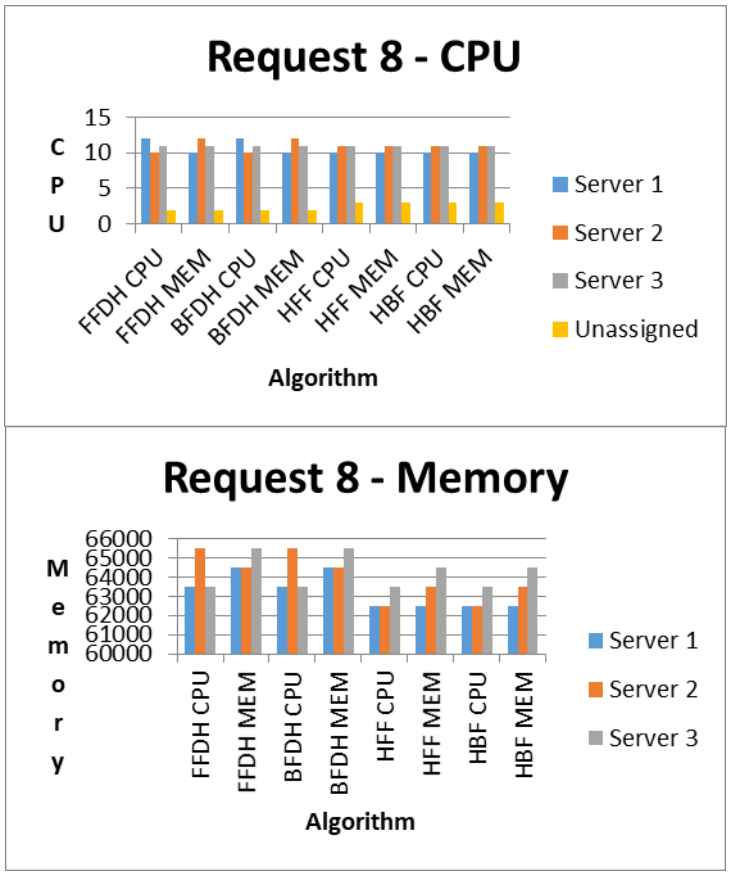

Figure 5.13. Results for eight set of requests.

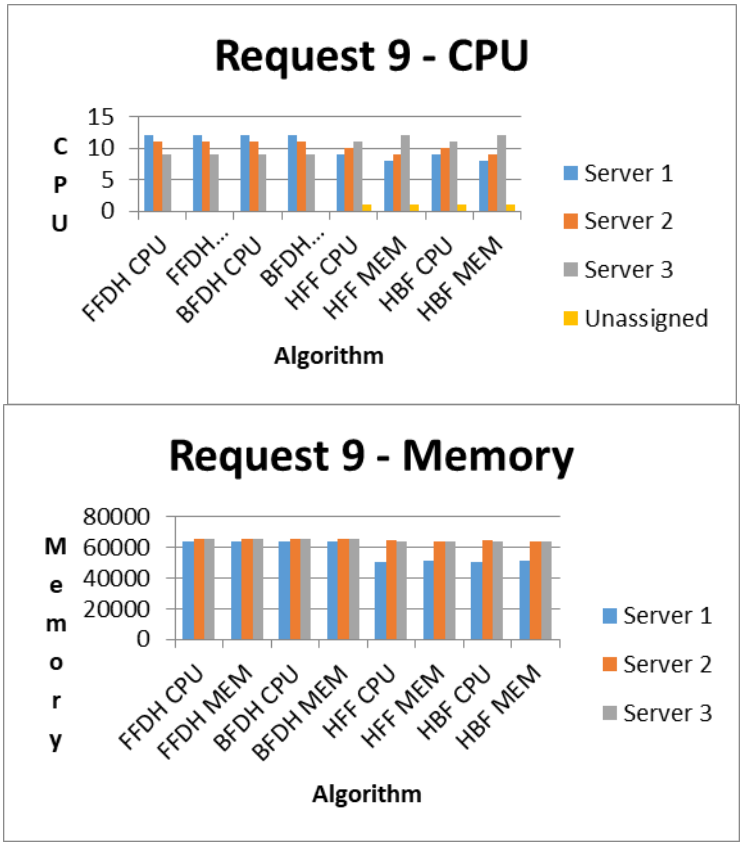

Figure 5.14. Results for ninth set of requests.

A qualitative analysis of these results yields the following observations:

- Given the nature of resource request dimensions in respect to server dimensions the results of FFDH and BFDH are similar.
- For both set of requests the method that provides the most balanced resource assignment are two-phased algorithms based on CPU.
- For both set of requests in respect of one-phase algorithms the ones that provide more balanced results are the ones that are based on CPU sorting.
- Using two-phase methods which divide server capacity result in less virtual machines being unassigned.
- Previous results are explained by the fact that sorting the request by Memory results in that both memory and CPU are ordered descending since in cloud computing high memory is associated with high amount of CPUs. While sorting by CPU results in a more heterogeneous sets since there are many flavors consisting of 1 CPU but memory ranging from 512 MB to 6GB; and this non-uniformity results in better resource assignment.
- A request which is memory saturated is harder to fit, since the magnitude of memory variations among flavors is bigger than CPU hops.
- The nature of the results obtained in resource assignment problems will depend largely on the proportions of CPU and memory contained in the

request and the CPU to memory ratio available on the servers, therefore two methods for efficient resource allocation are proposed:

- to perform and evaluate each algorithm for every request, but this can take time and resources in big data-center environments;
- or to first analyze the nature of the request and only performing the algorithm to be known most efficient for the type of request performed.
- For these type of requests where a high memory VM is associated with a high CPU count the best algorithm to use would be two-phase sorted by CPU count.

CONCLUSIONS

This section provides the conclusions on the project's research, implementation, activities and shortcomings; it also includes a consideration on NFV and SDN sustainability effects, and a list of possible related topics on which further work could be done.

6.1. Conclusions

An NFV scenario on physical infrastructure has been successfully deployed, in order to do this an operating Openstack Cloud was initially installed, later configuring telemetry services on it and finally instantiating an OSMANO based orchestrator for the deployment of network services. The guidelines for doing this were taken based on the reference provided by the Open Platform for NFV open-source project.

The installation procedures for Openstack as well as some services as Ceilometer prove to be a troublesome and complicated process since the official documentation does not tend to be completely clear and the troubleshooting requires time and considerable expertise operating and managing an Openstack environment. These installation procedures hindered the final development of the project since time was lost troubleshooting the installation and correct configuration of cloud components.

By correctly configuring the previously mentioned components some of the main components specified in ETSI NFV architecture were deployed, having the Network Function Virtualization Infrastructure, the Virtual Infrastructure Manager, the Virtual Network Function Manager and the Orchestrator. These elements are enough for successfully deploying a proof of concept scenario where you use the orchestrator for establishing network services and virtual network functions that are deployed in the Virtual Infrastructure Manager through API requests.

Doing such proof of concept allows to determine which functionalities have been actually implemented in the orchestrator in order to know how advanced the development actually is and where is room for introducing new features. A source code analysis was also done in order to determine how does the orchestrator interacts with the Virtual Infrastructure Manager and to obtain technical details regarding information handling; this was paramount in order to know what resources are actually required by the orchestrator in order to have a successful integration with other components. This last activity requires knowledge of the programming language used by the project as well as familiarity with the Virtual Infrastructure Manager APIs in order to interpret how information is handled between both components.

During the development of this project the latest release of Open Source MANO was released "release THREE", which introduced some new functionalities

being the most relevant to our field of study an experimental monitoring module which aims to collect telemetry data from the Virtual Infrastructure Manager. This new module overlaps with some of the work presented in this thesis. After deploying and interacting with OSM it is important to note that in order to effectively use this project an advanced knowledge of operation and management regarding the platform used as Virtual Infrastructure Manager is paramount; since proper configuration is needed between.

OSM installation proved to be troublesome at times, since trying to deploy the software on identical virtual machines with identical conditions yielded different results at different times without particular causes. Sometimes the installation process proved to be unsuccessful; other times some components where not correctly configured or deployed making the system incomplete and other times it deployed correctly. This indicates that the project might still be lacking some maturity and debugging.

The correct integration and configuration among components required to deploy an SDN/NFV environment is yet to be clearly documented and is not yet a fully standardized subject. Thorough examples or use-cases configuration templates using an orchestrator, Openstack as the Virtual Infrastructure Manager and an SDN controller are scarce; the most complete reference is OPNFV project which at the same time is not trivial to deploy and understand. This subject is still in an evolutionary phase, given that the configuration between the orchestrator, the networking-sfc project and the SDN controller can be done in several ways; depending on the networking back-end to be used.

Actually there are two ways to implement SFC in Openstack environments, using networking-sfc as the backend driver or using ODL. They differ in their implementation in the encapsulation method used since the first encapsulates SFC headers in MPLS and the latter uses NSH. This incongruence is caused by the fact that NSH has not been officially included in the mainstream release of OpenvSwitch, but will eventually disappear when NSH support is available.

It is important to note that the modules developed in this project where conceived as an introduction to this subject and where coded as stand-alone modules in order to initially approach the problem. To be able to incorporate it as a module into an orchestrator, its code has to be modified to fit the data models used in the project.

6.2. Environmental Impact

The most relevant environmental promise of the NFV paradigm is a more efficient use of hardware resources in operator’s environment, achieving this thorough the consolidation of workloads in servers and avoiding the use of hardware-appliances by consolidating their functions as VNFs. This approach provides some energy benefits:

- Reduction of the amount of physical devices deployed.
- Consolidation of workloads will require less energy to operate.

- Consolidation of workloads will require less space to operate and less cooling resources.
- The central orchestration of resources will enable the introduction of centralized energy-efficient policies in order to make a more efficient use of resources.

6.3. Future Lines of Study

Network Function Virtualization and Software Defined Networking are still in early evolutionary stages, therefore a lot of changes are still expected and there are many possibilities for future work.

In a practical approach the latest release of OSM has been already published, therefore it's installation along with the validation of new functionalities introduced is an interesting topic. Further work should include an extensive report on the information models it uses so new modules or features can be introduced. As the technology matures it is important to try and deploy more complex scenarios and validate the operational maturity of its components.

When Network Service Headers is merged into the upstream release of OpenvSwitch the implementation of a fully operational environment would be more friendly; as a proposal this could be made of an Openstack deployment configured with networking-sfc and OpenDaylight as Neutron backend, OVS with NSH support, OSM as the orchestrator and an ODL controller in charge of network configuration and forwarding rules. Such an environment would represent a fully functional reference NFV architecture.

Containers portability and lighter resource requirements have caused them to gain popularity as a possible way to deploy Virtual Network Functions in the future, therefore practical implementations of OpenStack scenarios integrated with Kubernetes or other container-orchestration platforms would be relevant.

As a direct continuation of this project, or engaging very similar topics:

- The integration of the developed modules into an orchestrator and Openstack working in a production environment.
- Further develop the resource assignment algorithms in order to take into account new relevant parameters as time constraints, quality of service considerations, hop count among service chains, among other options.
- Develop theoretical modules that address the resource optimization problem taking into account all the real-life variables that have to be taken into account in real implementations like efficient network or energy usage, space considerations for placing workloads, resource scalability and service growth.
- Evaluate VNF performance in deployed chains in production environments as well as resource consumption on bare-metal hosts.
- Develop high-availability scenarios that address the need of having multiple controllers and orchestrators serving as backups for production environments as well as multi-site deployment considerations.

- Conduct a study on energy savings that can be made by implementing different resource optimization assignment algorithms in NFV environments.

6.4. Ethical Considerations

SDN and NFV development have the potential to make a substantial change in telecommunications financial environment since it promises a comprehensive change in the approach by which networks are built. Taking into account modern trends where technology is embedded in every part of human life this can potentially affect general economy, lowering the prices for communication services worldwide by making a more efficient use of resources. Although this might be used by operators or IT providers to have a broader income for their investment and services; the way this will be brought into the economy is still to be defined.

The new architecture also brings a new depth to telco security since critical traffic will traverse one common platform, therefore a lot of sensitive information will be hosted in this platforms. This will require strict security enforcements in order to take care of customer's information privacy.

ACRONYMS

ACL	Access Control List
API	Application Programming Interface
ARP	Address Resolution Protocol
AWS	Amazon Web Services
BFD	Best Fit Decreasing
BFDH	Best Fit Decreasing Height
CFN	Cloud Formation
CLI	Command Line Interface
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name Server
ETH	Ethernet
ETSI	European Telecommunication Standards Institute
FBL	Finite Bottom Left
FFD	First Fit Decreasing
FFDH	First Fit Decreasing Height
FFF	Finite First Fit
FNF	Finite Next Fit
GB	Gigabyte
GRE	Generic Routing Encapsulation
GUI	Graphical User Interface
HBF	Hybrid Best Fit
HFF	Hybrid First Fit
HNF	Hybrid Next Fit
HOT	Heat Orchestration Template
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISG	Industry Standard Group
IT	Information Technology
JSON	Javascript Object Notation
KVM	Kernel-based Virtual Machine
LAN	Local Area Network
LXC	Linux Containers
MAC	Medium Access Control
MANO	Management and Orchestration
MB	Megabyte
MD-SAL	Model-Driven Service Abstraction Layer
MPLS	Multiprotocol Label Switching
NAT	Network Address Translation
NBL	Next-Bottom Left
NF	Network Functions
NFDH	Next-fit Descending Height
NFV	Network Functions Virtualization

NFVI	Network Function Virtualization Infrastructure
NS	Network Service
NSH	Network Service Headers
NTP	Network Time Protocol
ODL	OpenDaylight
ONOS	Open Network Operating System
OPNFV	Open Platform for NFV
OSI	Open System Interconnection
OSM	Open Source MANO
OSS/BSS	Operations/Business Support Systems
OVS	OpenvSwitch
PXE	Pre-boot Execution Environment
RAM	Random Access Memory
REST	Representational State Transfer
RO	Resource Orchestrator
SDN	Software-Defined Networking
SF	Service Function
SFC	Service Function Chaining
SFF	Service Function Forwarder
SFP	Service Function Path
SO	Service Orchestrator
SPI	Service Path Index
SQL	Structured Query Language
SRIOV	Single root input/output virtualization
UI	User Interface
UML	Unified Modelling Language
URL	Uniform Resource Locator
VCA	VNF Configuration Abstraction
VIM	Virtual Infrastructure Manager
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNF	Virtual Network Function
VNFFG	Virtual Network Function Forwarding Graph
VPN	Virtual Private Network
VXLAN	Virtual Extensible LAN
YAML	YAML Ain't Markup Language

REFERENCES

[1] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, “Network function virtualization: Challenges and opportunities for innovations,” IEEE Commun. Mag., vol. 53, no. 2, pp. 90–97, Feb. 2015.

[2] R. Guerzoni "Network functions virtualisation: An introduction, benefits, enables, challenges and call for action. Introductory white paper" in Proc. SDN OpenFlow World Congr., Jun 2012, pp. 1-16.

[3] "ETSI GS NFV 002 V1.2.1: Network Functions Virtualisation (NFV); Architectural framework," ETSI Ind. Spec. Group (ISG) Netw. Functions Virtualisation (NFV), Sophia-Antipolis Cedex, France, Dec. 2014. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/GS_NFV002v010201p.pdf

[4] “Software-Defined Networking Definition.” [Online]. Available: https://www.opennetworking.org/sdn-resources/sdn-definition [Accessed: 15-Jan-2017]

[5] “OpenFlow-enabled SDN and Network Functions Virtualization,” 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn

[6] M. Peter and G. Timothy, “The [NIST definition of cloud computing, recommendations of the National Institute of Standards and Technology,” Nat. Inst. Standards Technol. (NIST), Gaithersburg, MD, USA, Tech. Rep., Sep. 2011. [Online]. Available: http://www.nist.gov/itl/cloud/

[7] R. Mijumbi, J. Serrat, N. Bouten et al. “Network Function Virtualization: State-of-the-Art and Research Challenges,”IEEE COMMUNICATIONS, SURVEYS & TUTORIALS, vol. 18, No. 1, First Quarter 2016.

[8] P. Quinn and T. Nadeau, “RFC7498 - Problem Statement for Service Function Chaining,” Request for Comments (RFC) Pages - IETF, 2015. [Online]. Available: https://tools.ietf.org/html/rfc7498. [Accessed: 15-Jan-2017].

[9] J. Halpern and C. Pignataro, “RFC7665 - Service Function Chaining (SFC) Architecture,” IETF Datatracker, 2015. [Online]. Available: https://tools.ietf.org/html/rfc7665

[10] P. Quinn and U. Elzur, “Network Service Header,” 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-sfc-nsh-10.

[11] R. Guerzoni, “Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action. Introductory white paper,” in Proc. SDN OpenFlow World Congr., Jun. 2012 pp. 1–16.

[12] “ETSI - European Telecommunications Standards Institute – Networks Function Virtualisation.” [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/nfv [Accessed: 15-Jan-2017].

[13] “OpenStack - Open Source Cloud Computing Software.” [Online]. Available: https://www.openstack.org/ [Accessed: 15-Jan-2017]

[14] “OpenStack – Introduction to Openstack.” [Online]. Available: https://www.openstack.org/security-guide/introduction/introduction-to-openstack.html [Accessed: 15-Jan-2017]

[15] “OpenStack – Conceptual architecture.” [Online]. Available: https://docs.openstack.org/install-guide/get-started-conceptual-architecture.html [Accessed: 15-Jan-2017]

[16] “OpenStack – Openstack for telecom & NFV.” [Online]. Available: https://www.openstack.org/telecoms-and-nfv/ [Accessed: 15-Jan-2017]

[17] “OpenStack – Openstack Compute (nova).” [Online]. Available: https://docs.openstack.org/nova/latest/ [Accessed: 15-Jan-2017]

[18] “OpenStack – Nova System Architecture.” [Online]. Available: https://docs.openstack.org/nova/latest/user/architecture.html [Accessed: 15-Jan-2017]

[19] “OpenStack – Neutron Documentation.” [Online]. Available: https://docs.openstack.org/neutron/latest/ [Accessed: 15-Jan-2017]

[20] “OpenStack – Neutron ML2 plug-in.” [Online]. Available: https://docs.openstack.org/neutron/latest/admin/config-ml2.html [Accessed: 15-Jan-2017]

[21] “OpenStack – Service Function Chaining.” [Online]. Available: https://docs.openstack.org/neutron/latest/admin/config-sfc.html [Accessed: 15-Jan-2017]

[22] “OpenStack – Heat.” [Online]. Available: https://docs.openstack.org/heat/pike/ [Accessed: 15-Jan-2017]

[23] “OpenStack – Tacker.” [Online]. Available: https://docs.openstack.org/tacker/latest/ [Accessed: 15-Jan-2017]

[24] A. Vishnoi, T. Rozet and S. Hague, “OpenStack and OpenDaylight Integration for Service Function Chaining” 2016. [Online]. Available:

https://wiki.opendaylight.org/images/3/37/OpenDaylight-Summit-2016-OpenStack-SFC-Support.pdf [Accessed: 15-Jan-2017]

[25] "OpenStack – Ceilometer." [Online]. Available: https://docs.openstack.org/ceilometer/latest/ [Accessed: 15-Jan-2017]

[26] "The OpenDaylight platform." [Online]. Available: https://www.opendaylight.org/ [Accessed: 15-Jan-2017].

[27] "Apache Karaf." [Online]. Available: http://karaf.apache.org/. [Online]. [Accessed: 12-Feb-2017]

[28] "OpenDaylight controller – architectural framework." [Online]. Available: https://www.opendaylight.org/ [Accessed: 15-Jan-2017].

[29] "OpenDaylight controller – Cloud and NFV." [Online]. Available: https://www.opendaylight.org/use-cases-and-users/by-function/cloud-and-nfv [Accessed: 15-Jan-2017].

[30] "OpenVswitch." [Online]. Available: http://openvswitch.org/ [Online].[Accessed: 15-Jan-2017].

[31] I. Yamahata, "Opendaylight Summit - NetVirt Basic Tutorial." [Online]. Available: http://schd.ws/hosted_files/opendaylightsummit2016/c9/ODL Summit 2016 NetVirt Basic Tutorial %282%29.pdf. [Accessed: 10-Feb-2017].

[32] Yyang13, "Open vSwitch NSH patches." [Online]. Available: https://github.com/yyang13/ovs_nsh_patches [Accessed: 15-Jan-2017].

[33] A. Israel, A. Hoban, A. Tierno et al. "OSM Release Three – A Technical Overview" [Online]. Available: [Accessed: 15-Jan-2017].

[34] "JUJU" [Online]. Available: [Accessed: 15-Jan-2017].

[35] "Apache Kafka" [Online]. Available: kafka.apache.org/ [Accessed: 15-Jan-2017].

[36] "OSM Release THREE" [Online]. Available: https://osm.etsi.org/wikipub/index.php/OSM_Release_THREE [Accessed: 15-Jan-2017].

[37] "Linux Containers – LXD" [Online]. Available: [Accessed: 15-Jan-2017].

[38] "Openstack - Compute Schedulers" [Online]. Available: https://docs.openstack.org/ocata/config-reference/compute/schedulers.html#weights [Accessed: 15-Jan-2017].

[39] D. Gupta “Bin Packing Problem” [Online]. Available: https://www.geeksforgeeks.org/bin-packing-problem-minimize-number-of-used-bins/ [Accessed: 15-Jan-2017].

[40] http://cgi.csc.liv.ac.uk/~epa/surveyhtml.html#toc.2

[41] E.G. Coffman Jr. and M.R. Garey D.S. Johnson and R.E. Tarjan,“Performance bounds for level-oriented two-dimensional packing algorithms” *SIAM Journal on Computing*, 9:808--826, 1980.

[42] F.K.R. Chung, M.R. Garey, and D.S. Johnson,“On packing two-dimensional bins,” *SIAM J. Algebraic Discrete Methods*, 3:66--76, 1982.

[43] J.B. Frenk and G.G. Galambos,“Hybrid next-fit algorithm for the two-dimensional rectangle bin-packing problem,” *Computing*, No. 39 pp 201--217, 1987.

[44] J.O. Berkey and P.Y. Wong,“Two dimensional finite bin packing algorithms,” *Journal of Operational Research* Society, No.2 pp 423--429, 1987.

[45] L. Caccetta andA. Kulanoot. (2001). "Computational Aspects of Hard Knapsack Problems," *Nonlinear Analysis*. No. 47 pp 5547–5558.

[46] “Fuel” [Online]. Available: https://wiki.openstack.org/wiki/Fuel [Accessed: 15-Jan-2017].

[47] “Ceilometer Installation Guide [Online]. https://docs.openstack.org/mitaka/install-guide-ubuntu/ceilometer-install.html[Accessed: 15-Jan-2017].

[48] “OSM Release TWO” [Online]. Available: https://osm.etsi.org/wikipub/index.php/OSM_Release_TWO [Accessed: 15-Jan-2017].

ANNEX A. DETAILED SCENARIO SETUP GUIDE

A.1. Openstack Deployment

Openstack can be deployed in multiple configurations depending on the requirements of the implementation and hardware available. For this setup the objective was to have a simple setup that enables proof of concepts using basic functionality.

In our case we leveraged a tool for deploying Openstack in an automated manner using a tool called Fuel [46] which aids in the deploying and testing of Openstack clouds. Since Openstack uses several interfaces for communication of internal services, networking equipment has to configured accordingly to the networking parameters established in fuel deployment in order to get it working. The logical set-up of devices is described in the following image:

Fuel Openstack Configuration and Deployment

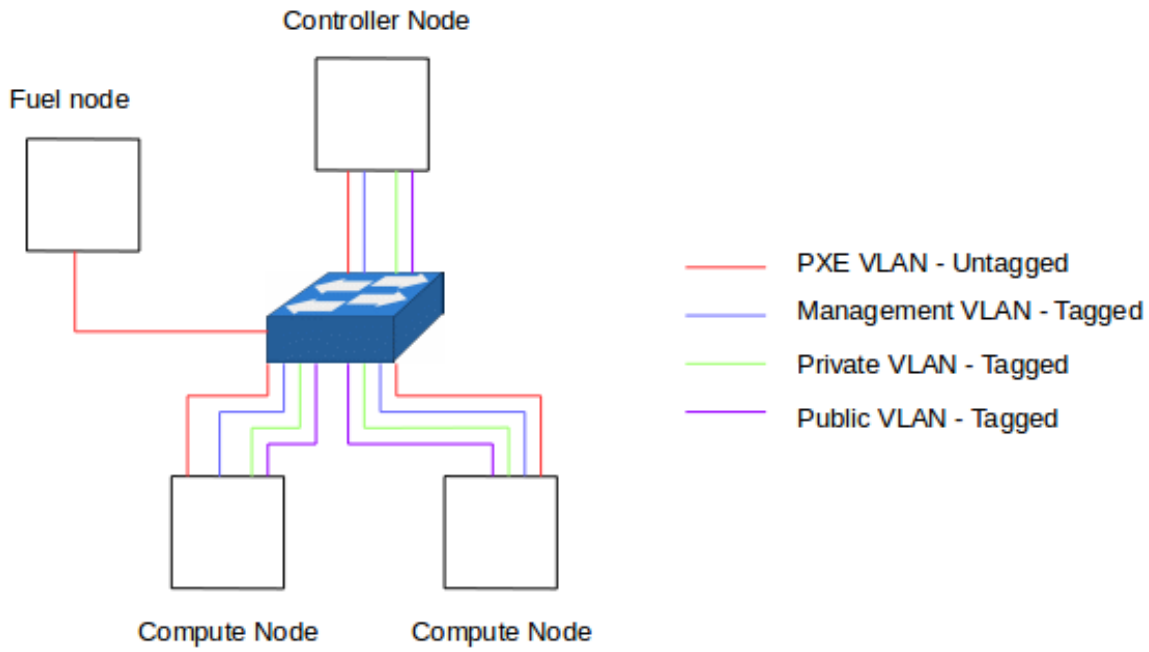

Figure A.1. Laboratory physical device set-up.

- All servers must be configured to boot through PXE and Fuel node server has to be configured to be in the same layer-2 network as the Openstack nodes so it can manage them. Once associated with the master node the following parameters are configured:
- Go to node tab an assign a role to each node. In this case one node is assigned as Controller and the remaining are assigned as Compute and Ceph. In this tab the interfaces for each node must be configured according to the parameters established on networking equipment.

- On the networks tab the addressing scheme for each of the networks mentioned in figure A.1 must be defined, as well as the VLAN tag for each interface. Public network must match the physical network where Openstack will communicate with external devices. Basic parameters as DNS servers and NTP servers must be selected.
- On the settings tab the hypervisor type must be selected according to the capabilities of servers to be used, choosing between KVM or QEMU depending if the servers support hardware virtualization.
- The remaining settings were left as provided by default, but the system can be tweaked according to each use case.

Finally, Fuel provides a health check tab to verify that Openstack has been correctly deployed, it is recommended to perform this revision when the deployment has been completed.

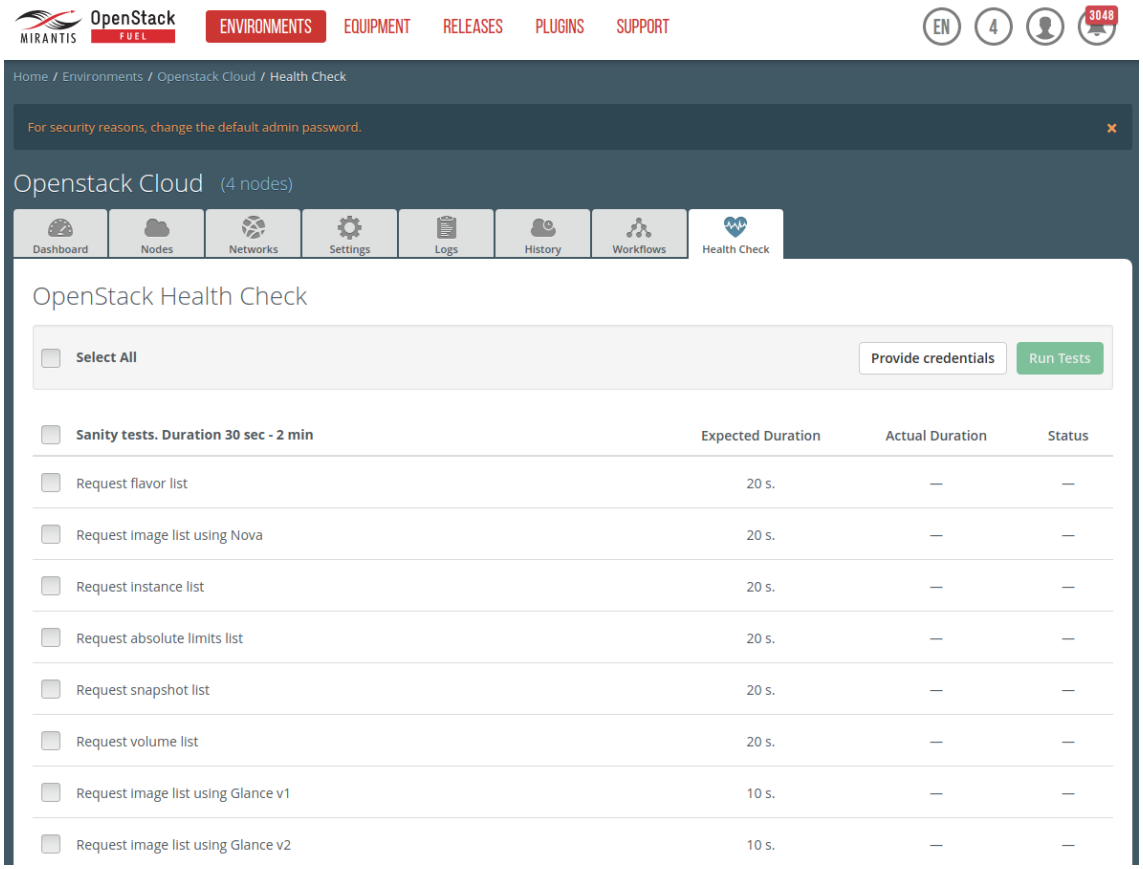

Figure A.2. Fuel health check feature through the Web UI.

Ceilometer Installation and Configuration

In this project Ceilometer service was installed manually following official Openstack documentation [47]. The method of procedure for this is the following:

- Install Mongo Database on the Controller node and set the binding IP address on the database configuration file to the node's management interface:

  sudo apt-get install mongodb-server mongodb-clients python-pymongo

- Create the user on the Database:

```
mongo --host localhost --eval '
db = db.getSiblingDB("ceilometer");
db.addUser({user: "ceilometer",
pwd: "ceilometer",
roles: [ "readWrite", "dbAdmin" ]})'
```

- Create the service in Openstack's database, add the service role, create the user and the endpoints. For this the IP address assigned to the management interface must be used.

```
openstack user create --domain default --password-prompt ceilometer
openstack role add --project services --user ceilometer admin
openstack service create --name ceilometer --description "Telemetry" metering
openstack endpoint create --region RegionOne metering public http://192.168.0.1:8777
openstack endpoint create --region RegionOne metering internal
http://192.168.0.1:8777
openstack endpoint create --region RegionOne metering admin http://192.168.0.1:8777
```

- Install Ceilometer's components on the controller node.

  apt-get install ceilometer-api ceilometer-collector ceilometer-agent-central ceilometer-agent-notification python-ceilometerclient

- Modify the configuration file introducing the deployment configuration parameters: database connection parameters, controller node IP address, rabbit user and password, keystone authentication parameters and service credentials.

```
[database]
connection = mongodb://ceilometer:ceilometer@192.168.0.1:27017/ceilometer

[DEFAULT]
auth_strategy=keystone
rpc_backend = rabbit

[oslo_messaging_rabbit]

rabbit_host=192.168.0.1
rabbit_password=3FNlQhGbtA2jj6rk4GcshHyZ
rabbit_userid=nova

[keystone_authtoken]
auth_uri=http://192.168.0.4:5000/v3
region_name=RegionOne
memcached_servers=192.168.0.1:11211
auth_type=password
project_name=services
```

```
password=ceilometer
username=ceilometer
project_domain_name=Default
user_domain_name=Default
auth_url=http://192.168.0.4:35357

[service_credentials]
auth_type = password
auth_url = http://192.168.0.4:5000/v3
project_domain_name = Default
user_domain_name = Default
project_name = services
username = ceilometer
password = ceilometer
interface = internalURL
region_name = RegionOne
```

- Change apache configuration to indicate where to find Ceilometer's API scripts. The file is located at "'/etc/apache2/sites-available/ceilometer.conf" and the address established there must contain the scripts.

```
<VirtualHost *:8777>
WSGIDaemonProcess ceilometer-api processes=2 threads=10 user=ceilometer group=ceilometer display-name=%{GROUP}
WSGIProcessGroup ceilometer-api
WSGIScriptAlias / "/var/www/cgi-bin/ceilometer/app"
WSGIApplicationGroup %{GLOBAL}
ErrorLog /var/log/apache2/ceilometer_error.log
CustomLog /var/log/apache2/ceilometer_access.log combined
</VirtualHost>

WSGISocketPrefix /var/run/apache2
```

- Install Ceilometer service in every compute node and modify the configuration file on each one in the same way as done in the controller.

```
apt-get install ceilometer-agent-compute
```

- Modify the configuration file for each service that wants to be monitored to enable notifications, the following example is for Neutron.

```
[oslo_messaging_notifications]
...
driver = messagingv2
```

- After modifications have been done to services all Ceilometer related services need to be restarted, as well as if any other service configuration file has been modified.

- To modify what information is to be collected by Ceilometer the file to be configured is located at "/etc/ceilometer/pipeline.yaml".

A.2. Open Source MANO installation and Configuration

Open Source MANO has the following requirements to be correctly installed [48]: have IP connectivity to the VIM and to the VNFs to be deployed, and comply with the server resource requirements which are 8 CPUs, 16 GB RAM, 100 GB disk and Ubuntu 16.04 configured to run LXD containers. Once these conditions have been met the following commands are to be issued:

```
wget https://osm-download.etsi.org/ftp/osm-2.0-two/install_from_source.sh
chmod +x install_from_source.sh
./install_from_source.sh -b tags/v2.0.2
```

Once the installation has been conducted successfully you should see as the final output in the screen:

```
{“success”:””}{“success”:””}
DONE
```

When correctly deployed the user interface can be accessed via a web browser, Google Chrome is recommended. The connection is done using https and port 8443. The default user and password for accessing this are both “admin”. Sometimes the system is not correctly configured at launch, it is important to check that in the accounts tab the OSMJUJU is registered and in the Config tab the resource orchestrator is also configured.

To configure OSM to work with Openstack as the VIM the following steps must be followed:

- Go into the Resource Orchestrator and register the VIM with the information required for API access which is the IP address and port for keystone service and user credentials (additional information like security groups can be specified, but the aforementioned are the required parameters).

  ```
  lxc exec RO -- bash

  export OPENMANO_TENANT=osm
  openmano datacenter-create openstack-site http://192.168.88.32:5000/v2.0 --type openstack --description "OpenStack site"
  openmano datacenter-attach openstack-site --user=admin --password=userpwd --vim-tenant-name=admin
  openmano datacenter-list
  exit     #or Ctrl+D to get out of the RO container
  ```

- For correct operation between OSM and Openstack is recommended to create a provider network on Openstack that will be a management network for VNFs.

```
neutron net-create mgmt --provider:network_type=vlan --
provider:physical_network=physnet_em1 --provider:segmentation_id=500 --shared
neutron subnet-create --name subnet-mgmt mgmt 10.208.0.0/24 --allocation-pool
start=10.208.0.2, end=10.208.0.254
```

- It is recommended that the images to be used by VNFs instantiated via OSM to be previously declared since the image creation process tends to take some time

- In order to validate the correct configuration of the VIM in OSM a network service can be deployed to validate that it is correctly instantiated in Openstack. For this the network service and the VNFs involved should be previously on boarded.

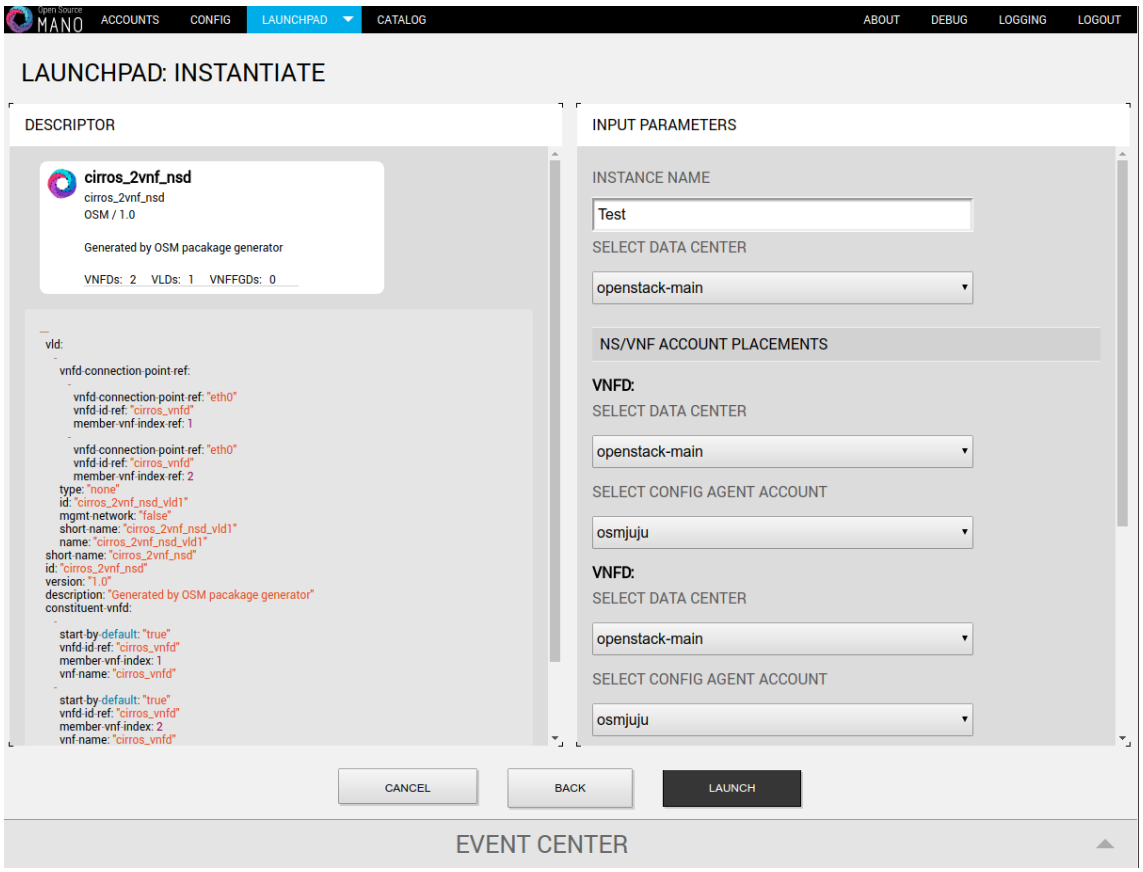

Figure A.3. Open Source MANO Network Service Instantiation.

ANNEX B. OPENSTACK RESOURCE COLLECTION FRAMEWORK

In this section the steps followed to query Openstack Telemetry services and the consequent storage of this information in a database will be described.

B.1. Openstack API Access

In order to access Openstack API access two sets of information must be gathered; on the first-hand the API addresses must be defined. In order to do so access to the controller node is required; an example of how to get the API endpoints for some services follows:

- List endpoints.

```
root@node-1:~# openstack endpoint list
+----------------------------------+-----------+---------------+----------------+
| ID                               | Region    | Service Name  | Service Type   |
+----------------------------------+-----------+---------------+----------------+
| 28afc605f0cf492f8251e749f053c804 | RegionOne | ceilometer    | metering       |
| 638d692ddf3c415d8c0a0b3bfd9292c6 | RegionOne | cinder        | volume         |
| 4e19cf14698a456297a678188f1f4ec2 | RegionOne | compute_legacy | compute_legacy |
| e5b73268d17e43eaaac9d5e300d34c9f | RegionOne | glance        | image          |
| b54a9926c3724d2781f47bc7c1565484 | RegionOne | nova          | compute        |
| 235e1b9021ea45c19c9263cedee8b973 | RegionOne | swift         | object-store   |
| f7565b0bce374d8d9e1420ce7bae918e | RegionOne | neutron       | network        |
| 21dc72c95d264ae68387d0eb689cac96 | RegionOne | heat          | orchestration  |
| f0949456a4cc4801b5def1cc778973bd | RegionOne | cinderv3      | volumev3       |
| d390af32fd3e480c8b1c7b75556346fb | RegionOne | keystone      | identity       |
| 0a486c2fdcb2462faf238695206d0787 | RegionOne | aodh          | alarming       |
| 0b5add5d7c8c4862b7ede3a9e6057d8a | RegionOne | cinderv2      | volumev2       |
| 7b9b534b231544c5ab06021e165944c5 | RegionOne | glare         | artifact       |
| 4df54188480d44aa855958fc44724a9f | RegionOne | heat-cfn      | cloudformation |
| d30b86ccadd048a997f3d29bcbc4fb3c | RegionOne | swift_s3      | s3             |
+----------------------------------+-----------+---------------+----------------+
```

- Get the details related to the endpoint to be used:

```
root@node-1:~# openstack endpoint show 28afc605f0cf492f8251e749f053c804
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| adminurl     | http://192.168.0.6:8777          |
| enabled      | True                             |
| id           | 28afc605f0cf492f8251e749f053c804 |
| internalurl  | http://192.168.0.6:8777          |
| publicurl    | http://192.168.88.32:8777        |
| region       | RegionOne                        |
| service_id   | 61a659a638734176963358f419bfe587 |
| service_name | ceilometer                       |
| service_type | metering                         |
+--------------+----------------------------------+
```

- In order for any script to interact with Openstack APIs it must have IP connectivity to one of the endpoints specified. Best-practices suggest that administrative tasks should be performed over the secured admin VLAN.

It is also required, that scripts establish a connection to Openstack APIs by providing authentication information which is used to obtain a token used for each session. These authentication parameters can be either downloaded from the Dashboard or copied from the controller. An example of how to do it through the dashboard follows:

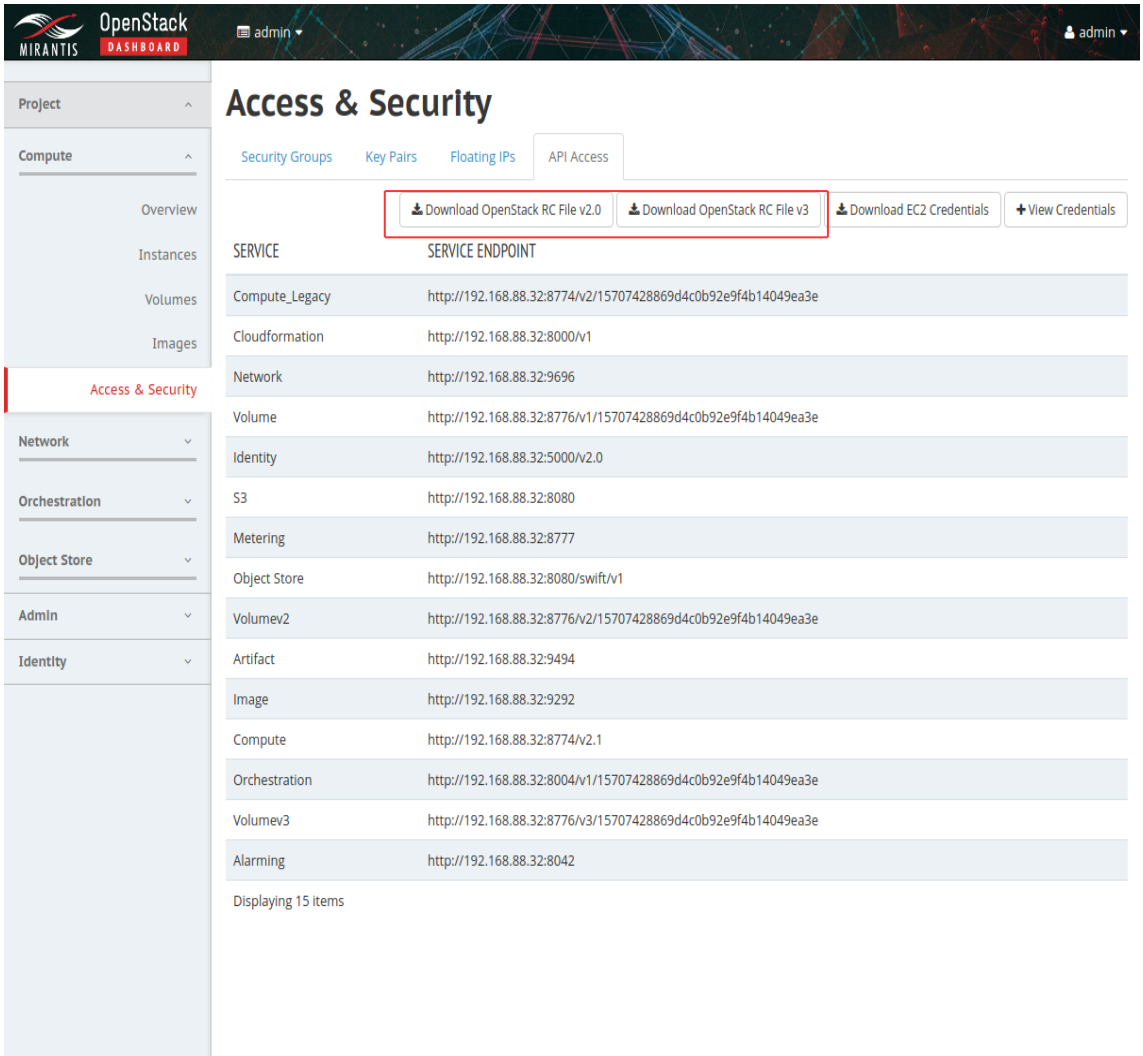

Figure B.1. OpenStack API Credentials Location.

Once the file containing these credentials is located at the device from which the API connection is to be established the best-practice is to set this variables as a environment variables and have the script get them every time for each connection. This is a security measure since hard-coding credentials into scripts is not recommended:

- Find the file in the working directory (The filename usually contains “openrc”):

```
root@node-1:~# ls
ceph.bootstrap-mds.keyring  ceph.client.admin.keyring  ceph.log        mininet-vm.img
SDN_Tut_Sus.img
ceph.bootstrap-osd.keyring  ceph.conf                  ceph.mon.keyring  openrc
```

- Set the information as environment variables.

```
root@node-1:~# source openrc
```

In order for the scripts to work correctly Openstack must be providing all the resource measures they are programmed to get, therefore before executing them this must be verified. This can be done by executing Ceilometer commands within the controller node:

```
root@node-43:~# ceilometer meter-list
+--------------------------------+------------+------------+------------------------------------------------
-----------------+----------------------------------+----------------------------------+
| Name                           | Type       | Unit       | Resource ID
| User ID                        | Project ID                       |
+--------------------------------+------------+------------+------------------------------------------------
-----------------+----------------------------------+----------------------------------+
| compute.instance.booting.time  | gauge      | sec        | 67e27c70-f12a-46ea-9b26-
4503a3d77524                     | 932d739b610c44ee911874352a0de05e |
17ec7d27028c44c0baf3d5f4b2f7ed15 |
| cpu                            | cumulative | ns         | 67e27c70-f12a-46ea-9b26-
4503a3d77524                     | 932d739b610c44ee911874352a0de05e |
17ec7d27028c44c0baf3d5f4b2f7ed15 |
| cpu.delta                      | delta      | ns         | 67e27c70-f12a-46ea-9b26-4503a3d77524
| 932d739b610c44ee911874352a0de05e | 17ec7d27028c44c0baf3d5f4b2f7ed15 |
| cpu_util                       | gauge      | %          | 67e27c70-f12a-46ea-9b26-
4503a3d77524                     | 932d739b610c44ee911874352a0de05e |
17ec7d27028c44c0baf3d5f4b2f7ed15 |
| disk.allocation                | gauge      | B          | 67e27c70-f12a-46ea-9b26-
4503a3d77524                     | 932d739b610c44ee911874352a0de05e |
17ec7d27028c44c0baf3d5f4b2f7ed15 |
| disk.capacity                  | gauge      | B          | 67e27c70-f12a-46ea-9b26-
4503a3d77524                     | 932d739b610c44ee911874352a0de05e |
17ec7d27028c44c0baf3d5f4b2f7ed15 |
| disk.device.allocation         | gauge      | B          | 67e27c70-f12a-46ea-9b26-
4503a3d77524-vda                 | 932d739b610c44ee911874352a0de05e |
17ec7d27028c44c0baf3d5f4b2f7ed15 |
| disk.device.allocation         | gauge      | B          | 67e27c70-f12a-46ea-9b26-
4503a3d77524-vdb                 | 932d739b610c44ee911874352a0de05e |
17ec7d27028c44c0baf3d5f4b2f7ed15 |
| disk.device.capacity           | gauge      | B          | 67e27c70-f12a-46ea-9b26-
4503a3d77524-vda                 | 932d739b610c44ee911874352a0de05e |
17ec7d27028c44c0baf3d5f4b2f7ed15 |
| disk.device.capacity           | gauge      | B          | 67e27c70-f12a-46ea-9b26-
4503a3d77524-vdb                 | 932d739b610c44ee911874352a0de05e |
17ec7d27028c44c0baf3d5f4b2f7ed15 |
| disk.device.read.bytes         | cumulative | B          | 67e27c70-f12a-46ea-9b26-
4503a3d77524-vda                 | 932d739b610c44ee911874352a0de05e |
17ec7d27028c44c0baf3d5f4b2f7ed15 |
```

disk.device.read.bytes	cumulative	B	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.read.bytes.rate	gauge	B/s	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.read.bytes.rate	gauge	B/s	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.read.requests	cumulative	request	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.read.requests	cumulative	request	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.read.requests.rate	gauge	request/s	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.read.requests.rate	gauge	request/s	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.read.requests.rate	gauge	requests/s	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.read.requests.rate	gauge	requests/s	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.usage	gauge	B	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.usage	gauge	B	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.write.bytes	cumulative	B	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.write.bytes	cumulative	B	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.write.bytes.rate	gauge	B/s	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.write.bytes.rate	gauge	B/s	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.write.requests	cumulative	request	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.write.requests	cumulative	request	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.write.requests.rate	gauge	request/s	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.write.requests.rate	gauge	request/s	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.device.write.requests.rate	gauge	requests/s	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15

disk.device.write.requests.rate	gauge	requests/s	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.ephemeral.size	gauge	GB	67e27c70-f12a-46ea-9b26-4503a3d77524	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.read.bytes	cumulative	B	67e27c70-f12a-46ea-9b26-4503a3d77524	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.read.bytes.rate	gauge	B/s	67e27c70-f12a-46ea-9b26-4503a3d77524	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.read.requests	cumulative	request	67e27c70-f12a-46ea-9b26-4503a3d77524	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.read.requests.rate	gauge	request/s	67e27c70-f12a-46ea-9b26-4503a3d77524	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.read.requests.rate	gauge	requests/s	67e27c70-f12a-46ea-9b26-4503a3d77524	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.root.size	gauge	GB	67e27c70-f12a-46ea-9b26-4503a3d77524	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.usage	gauge	B	67e27c70-f12a-46ea-9b26-4503a3d77524	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.write.bytes	cumulative	B	67e27c70-f12a-46ea-9b26-4503a3d77524	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.write.bytes.rate	gauge	B/s	67e27c70-f12a-46ea-9b26-4503a3d77524	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.write.requests	cumulative	request	67e27c70-f12a-46ea-9b26-4503a3d77524	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.write.requests.rate	gauge	request/s	67e27c70-f12a-46ea-9b26-4503a3d77524	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
disk.write.requests.rate	gauge	requests/s	67e27c70-f12a-46ea-9b26-4503a3d77524	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15
image	gauge	image	3f311f35-d67e-4d6c-928d-02ffb5c70afc	None	17ec7d27028c44c0baf3d5f4b2f7ed15
image	gauge	image	f1a578cb-3bb3-45d9-a632-48d7067e6aba	None	17ec7d27028c44c0baf3d5f4b2f7ed15
image.download	delta	B	3f311f35-d67e-4d6c-928d-02ffb5c70afc	None	17ec7d27028c44c0baf3d5f4b2f7ed15
image.serve	delta	B	3f311f35-d67e-4d6c-928d-02ffb5c70afc	None	17ec7d27028c44c0baf3d5f4b2f7ed15
image.size	gauge	B	3f311f35-d67e-4d6c-928d-02ffb5c70afc	None	17ec7d27028c44c0baf3d5f4b2f7ed15
image.size	gauge	B	f1a578cb-3bb3-45d9-a632-48d7067e6aba	None	17ec7d27028c44c0baf3d5f4b2f7ed15
instance	gauge	instance	67e27c70-f12a-46ea-9b26-4503a3d77524	932d739b610c44ee911874352a0de05e	17ec7d27028c44c0baf3d5f4b2f7ed15

```
| ip.floating                  | gauge      | ip         | b38a78be-b4b0-4bca-849e-2feed2fc2e14
| None                         | 17ec7d27028c44c0baf3d5f4b2f7ed15 |
| memory                       | gauge      | MB         | 67e27c70-f12a-46ea-9b26-
4503a3d77524                   | 932d739b610c44ee911874352a0de05e |
17ec7d27028c44c0baf3d5f4b2f7ed15 |
| memory.resident              | gauge      | MB         | 67e27c70-f12a-46ea-9b26-
4503a3d77524                   | 932d739b610c44ee911874352a0de05e |
17ec7d27028c44c0baf3d5f4b2f7ed15 |
| memory.usage                 | gauge      | MB         | 67e27c70-f12a-46ea-9b26-
4503a3d77524                   | 932d739b610c44ee911874352a0de05e |
17ec7d27028c44c0baf3d5f4b2f7ed15 |
| network.incoming.bytes       | cumulative | B          | instance-0000000f-67e27c70-
f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd |
932d739b610c44ee911874352a0de05e | 17ec7d27028c44c0baf3d5f4b2f7ed15 |
| network.incoming.bytes.rate  | gauge      | B/s        | instance-0000000f-67e27c70-
f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd |
932d739b610c44ee911874352a0de05e | 17ec7d27028c44c0baf3d5f4b2f7ed15 |
| network.incoming.packets     | cumulative | packet     | instance-0000000f-67e27c70-
f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd |
932d739b610c44ee911874352a0de05e | 17ec7d27028c44c0baf3d5f4b2f7ed15 |
| network.incoming.packets.rate | gauge     | packet/s   | instance-0000000f-67e27c70-
f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd |
932d739b610c44ee911874352a0de05e | 17ec7d27028c44c0baf3d5f4b2f7ed15 |
| network.outgoing.bytes       | cumulative | B          | instance-0000000f-67e27c70-
f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd |
932d739b610c44ee911874352a0de05e | 17ec7d27028c44c0baf3d5f4b2f7ed15 |
| network.outgoing.bytes.rate  | gauge      | B/s        | instance-0000000f-67e27c70-
f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd |
932d739b610c44ee911874352a0de05e | 17ec7d27028c44c0baf3d5f4b2f7ed15 |
| network.outgoing.packets     | cumulative | packet     | instance-0000000f-67e27c70-
f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd |
932d739b610c44ee911874352a0de05e | 17ec7d27028c44c0baf3d5f4b2f7ed15 |
| network.outgoing.packets.rate | gauge     | packet/s   | instance-0000000f-67e27c70-
f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd |
932d739b610c44ee911874352a0de05e | 17ec7d27028c44c0baf3d5f4b2f7ed15 |
| vcpus                        | gauge      | vcpu       | 67e27c70-f12a-46ea-9b26-
4503a3d77524                   | 932d739b610c44ee911874352a0de05e |
17ec7d27028c44c0baf3d5f4b2f7ed15 |
+------------------------------+------------+------------+-------------------------------------------------
------------------+----------------------------------+----------------------------------+

root@node-43:~# ceilometer resource-list
+--------------------------------------------------------------------------+-----------+-----------------------------
-----+----------------------------------+
| Resource ID                                                              | Source    | User ID                      |
Project ID                       |
+--------------------------------------------------------------------------+-----------+-----------------------------
-----+----------------------------------+
| 3f311f35-d67e-4d6c-928d-02ffb5c70afc                                     | openstack | None
| 17ec7d27028c44c0baf3d5f4b2f7ed15 |
| 67e27c70-f12a-46ea-9b26-4503a3d77524                                       | openstack |
932d739b610c44ee911874352a0de05e | 17ec7d27028c44c0baf3d5f4b2f7ed15 |
| 67e27c70-f12a-46ea-9b26-4503a3d77524-vda                                   | openstack |
932d739b610c44ee911874352a0de05e | 17ec7d27028c44c0baf3d5f4b2f7ed15 |
| 67e27c70-f12a-46ea-9b26-4503a3d77524-vdb                                   | openstack |
932d739b610c44ee911874352a0de05e | 17ec7d27028c44c0baf3d5f4b2f7ed15 |
| b38a78be-b4b0-4bca-849e-2feed2fc2e14                                       | openstack | None
| 17ec7d27028c44c0baf3d5f4b2f7ed15 |
| f1a578cb-3bb3-45d9-a632-48d7067e6aba                                       | openstack | None
| 17ec7d27028c44c0baf3d5f4b2f7ed15 |
```

```
| instance-0000000f-67e27c70-f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd |
openstack | 932d739b610c44ee911874352a0de05e |
17ec7d27028c44c0baf3d5f4b2f7ed15 |
+-------------------------------------------------------------------------+----------+-----------------------------
-----+---------------------------------+

root@node-43:~# ceilometer sample-list
+--------------------------------------+--------------------------------------------------------------------------+--
------------------------------+------------+------------------+------------+--------------------------+
| ID                                   | Resource ID                                                              | Name
| Type       | Volume           | Unit       | Timestamp                |
+--------------------------------------+--------------------------------------------------------------------------+--
------------------------------+------------+------------------+------------+--------------------------+
| 73b22905-ab27-11e7-80a2-00241d958c54 | 3f311f35-d67e-4d6c-928d-02ffb5c70afc
| image.size                 | gauge      | 322764800.0      | B          | 2017-10-
07T06:19:19.132000 |
| 73b22906-ab27-11e7-80a2-00241d958c54 | f1a578cb-3bb3-45d9-a632-
48d7067e6aba                       | image.size                 | gauge      | 23986176.0
| B          | 2017-10-07T06:19:19.132000 |
| 73b22904-ab27-11e7-80a2-00241d958c54 | b38a78be-b4b0-4bca-849e-2feed2fc2e14
| ip.floating                | gauge      | 1.0              | ip         | 2017-10-
07T06:19:19.121000 |
| 72abd4a6-ab27-11e7-80a2-00241d958c54 | 3f311f35-d67e-4d6c-928d-02ffb5c70afc
| image                      | gauge      | 1.0              | image      | 2017-10-
07T06:19:17.402000 |
| 72abd4a7-ab27-11e7-80a2-00241d958c54 | f1a578cb-3bb3-45d9-a632-
48d7067e6aba                       | image                      | gauge      | 1.0              |
image      | 2017-10-07T06:19:17.402000 |
| 505a8686-ab27-11e7-80a2-00241d958c54 | 67e27c70-f12a-46ea-9b26-
4503a3d77524                       | disk.read.requests.rate    | gauge      | 0.0
| request/s  | 2017-10-07T06:18:19.223000 |
| 4fde13dd-ab27-11e7-85a6-00241d958c52 | 67e27c70-f12a-46ea-9b26-4503a3d77524
| disk.read.requests         | cumulative | 21797.0          | request    | 2017-10-
07T06:18:19.223000 |
| 4fde13dc-ab27-11e7-85a6-00241d958c52 | 67e27c70-f12a-46ea-9b26-4503a3d77524
| disk.write.requests.rate   | gauge      | 0.068306032892   | requests/s | 2017-10-
07T06:18:19.218000 |
| 4fde13da-ab27-11e7-85a6-00241d958c52 | 67e27c70-f12a-46ea-9b26-
4503a3d77524-vda                   | disk.device.read.bytes.rate | gauge      | 0.0
| B/s        | 2017-10-07T06:18:19.212000 |
| 4fde13db-ab27-11e7-85a6-00241d958c52 | 67e27c70-f12a-46ea-9b26-
4503a3d77524-vdb                   | disk.device.read.bytes.rate | gauge      | 0.0
| B/s        | 2017-10-07T06:18:19.212000 |
| 504aa5d6-ab27-11e7-80a2-00241d958c54 | 67e27c70-f12a-46ea-9b26-
4503a3d77524-vda                   | disk.device.write.bytes.rate | gauge      | 0.0
| B/s        | 2017-10-07T06:18:19.207000 |
| 504aa5d7-ab27-11e7-80a2-00241d958c54 | 67e27c70-f12a-46ea-9b26-
4503a3d77524-vdb                   | disk.device.write.bytes.rate | gauge      | 0.0
| B/s        | 2017-10-07T06:18:19.207000 |
| 4fde13d8-ab27-11e7-85a6-00241d958c52 | 67e27c70-f12a-46ea-9b26-
4503a3d77524-vda                   | disk.device.write.bytes    | cumulative |
8060896256.0       | B          | 2017-10-07T06:18:19.207000 |
| 4fde13d9-ab27-11e7-85a6-00241d958c52 | 67e27c70-f12a-46ea-9b26-
4503a3d77524-vdb                   | disk.device.write.bytes    | cumulative | 0.0
| B          | 2017-10-07T06:18:19.207000 |
| 4fde13d6-ab27-11e7-85a6-00241d958c52 | 67e27c70-f12a-46ea-9b26-
4503a3d77524-vda                   | disk.device.write.requests.rate | gauge      |
0.068306032892     | requests/s | 2017-10-07T06:18:19.199000 |
```

4fde13d7-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.write.requests.rate	gauge	0.0	requests/s	2017-10-07T06:18:19.199000
4fde13d5-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	instance	gauge	1.0	instance	2017-10-07T06:18:19.194000
4fde13d3-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.write.bytes.rate	gauge	443.556053589	B/s	2017-10-07T06:18:19.189000
4fde13d4-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.write.bytes.rate	gauge	0.0	B/s	2017-10-07T06:18:19.189000
4fde13d1-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.usage	gauge	1.61061277696e+11	B	2017-10-07T06:18:19.183000
4fde13d2-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.usage	gauge	212992.0	B	2017-10-07T06:18:19.183000
4fde13d0-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	memory.usage	gauge	1446.0	MB	2017-10-07T06:18:19.175000
4fde13cf-ab27-11e7-85a6-00241d958c52	instance-0000000f-67e27c70-f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd	network.outgoing.bytes	cumulative	10310405.0	B	2017-10-07T06:18:19.169000
503a26ca-ab27-11e7-80a2-00241d958c54	instance-0000000f-67e27c70-f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd	network.outgoing.bytes.rate	gauge	0.0	B/s	2017-10-07T06:18:19.169000
4fde13ce-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	disk.write.bytes.rate	gauge	443.556053589	B/s	2017-10-07T06:18:19.165000
502ff39e-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.read.requests.rate	gauge	0.0	request/s	2017-10-07T06:18:19.157000
502ff39f-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.read.requests.rate	gauge	0.0	request/s	2017-10-07T06:18:19.157000
4fde13cc-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.read.requests	cumulative	21020.0	request	2017-10-07T06:18:19.157000
4fde13cd-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.read.requests	cumulative	777.0	request	2017-10-07T06:18:19.157000
502a82d8-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.write.requests.rate	gauge	0.0	request/s	2017-10-07T06:18:19.152000
502a82d9-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.write.requests.rate	gauge	0.0	request/s	2017-10-07T06:18:19.152000
4fde13ca-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.write.requests	cumulative	242110.0	request	2017-10-07T06:18:19.152000
4fde13cb-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.write.requests	cumulative	0.0	request	2017-10-07T06:18:19.152000
4fde13c9-ab27-11e7-85a6-00241d958c52	instance-0000000f-67e27c70-f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd	network.outgoing.bytes.rate	gauge	1.91921322813	B/s	2017-10-07T06:18:19.135000
4fde13c7-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.allocation	gauge	1932591104.0	B	2017-10-07T06:18:19.130000

4fde13c8-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.allocation	gauge	0.0	B	2017-10-07T06:18:19.130000
5021a8a2-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524	disk.write.requests.rate	gauge	0.0	request/s	2017-10-07T06:18:19.126000
4fde13c6-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	disk.write.requests	cumulative	242110.0	request	2017-10-07T06:18:19.126000
4fde13c5-ab27-11e7-85a6-00241d958c52	instance-0000000f-67e27c70-f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd	network.incoming.packets	cumulative	795087.0	packet	2017-10-07T06:18:19.122000
501f8eaa-ab27-11e7-80a2-00241d958c54	instance-0000000f-67e27c70-f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd	network.incoming.packets.rate	gauge	0.0	packet/s	2017-10-07T06:18:19.122000
4fde13c4-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	cpu_util	gauge	4.2132023319	%	2017-10-07T06:18:19.116000
4fde13c3-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	disk.allocation	gauge	1932591104.0	B	2017-10-07T06:18:19.111000
4fde13c2-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	memory.resident	gauge	3356.0	MB	2017-10-07T06:18:19.097000
4fde13c1-ab27-11e7-85a6-00241d958c52	instance-0000000f-67e27c70-f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd	network.incoming.bytes.rate	gauge	25.1696818148	B/s	2017-10-07T06:18:19.090000
4fde13c0-ab27-11e7-85a6-00241d958c52	instance-0000000f-67e27c70-f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd	network.incoming.bytes	cumulative	300162843.0	B	2017-10-07T06:18:19.085000
500ead7e-ab27-11e7-80a2-00241d958c54	instance-0000000f-67e27c70-f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd	network.incoming.bytes.rate	gauge	0.0	B/s	2017-10-07T06:18:19.085000
4fde13be-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.read.requests.rate	gauge	0.0	requests/s	2017-10-07T06:18:19.079000
4fde13bf-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.read.requests.rate	gauge	0.0	requests/s	2017-10-07T06:18:19.079000
4fde13bc-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.capacity	gauge	1.610612736e+11	B	2017-10-07T06:18:19.073000
4fde13bd-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.capacity	gauge	67108864.0	B	2017-10-07T06:18:19.073000
4fde13bb-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	disk.read.bytes.rate	gauge	0.0	B/s	2017-10-07T06:18:19.067000
4fde13ba-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	disk.read.requests.rate	gauge	0.0	requests/s	2017-10-07T06:18:19.058000
4fde13b9-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	disk.usage	gauge	1.61061490688e+11	B	2017-10-07T06:18:19.053000
4ffb823a-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524	disk.read.bytes.rate	gauge	0.0	B/s	2017-10-07T06:18:19.049000
4fde13b8-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	disk.read.bytes	cumulative	361184768.0	B	2017-10-07T06:18:19.049000

4fde13b7-ab27-11e7-85a6-00241d958c52	instance-0000000f-67e27c70-f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd	network.outgoing.packets	cumulative	75962.0	packet	2017-10-07T06:18:19.043000
4ffe3df4-ab27-11e7-80a2-00241d958c54	instance-0000000f-67e27c70-f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd	network.outgoing.packets.rate	gauge	0.0	packet/s	2017-10-07T06:18:19.043000
4fde13b6-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	disk.capacity	gauge	1.61128382464e+11	B	2017-10-07T06:18:19.032000
4fe5555a-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524	disk.write.bytes.rate	gauge	0.0	B/s	2017-10-07T06:18:19.027000
4fde13b5-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	disk.write.bytes	cumulative	8060896256.0	B	2017-10-07T06:18:19.027000
4fe2d802-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.read.bytes.rate	gauge	0.0	B/s	2017-10-07T06:18:19.017000
4fe2d803-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.read.bytes.rate	gauge	0.0	B/s	2017-10-07T06:18:19.017000
4fde13b3-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.read.bytes	cumulative	348197888.0	B	2017-10-07T06:18:19.017000
4fde13b4-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.read.bytes	cumulative	12986880.0	B	2017-10-07T06:18:19.017000
4fe09b00-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524	cpu.delta	delta	70000000.0	ns	2017-10-07T06:18:19.010000
4fde13b2-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	cpu	cumulative	1.5395543e+14	ns	2017-10-07T06:18:19.010000
4fe09b01-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524	cpu_util	gauge	6.2110643901	%	2017-10-07T06:18:19.010000
4f2c4905-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	disk.write.requests.rate	gauge	0.0683240852129	requests/s	2017-10-07T06:18:18.050000
4f2c4903-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.read.bytes.rate	gauge	0.0	B/s	2017-10-07T06:18:18.045000
4f2c4904-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.read.bytes.rate	gauge	0.0	B/s	2017-10-07T06:18:18.045000
4f2c4902-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	disk.capacity	gauge	1.61128382464e+11	B	2017-10-07T06:18:18.039000
4f2c4901-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	instance	gauge	1.0	instance	2017-10-07T06:18:18.035000
4f2c48ff-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.write.bytes.rate	gauge	443.673279197	B/s	2017-10-07T06:18:18.029000
4f2c4900-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.write.bytes.rate	gauge	0.0	B/s	2017-10-07T06:18:18.029000
4f2c48fd-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.usage	gauge	1.61061277696e+11	B	2017-10-07T06:18:18.024000

4f2c48fe-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.usage	gauge	212992.0	B	2017-10-07T06:18:18.024000
4f8b1018-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.read.bytes.rate	gauge	0.0	B/s	2017-10-07T06:18:18.017000
4f8b1019-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.read.bytes.rate	gauge	0.0	B/s	2017-10-07T06:18:18.017000
4f2c48fb-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.read.bytes	cumulative	348197888.0	B	2017-10-07T06:18:18.017000
4f2c48fc-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.read.bytes	cumulative	12986880.0	B	2017-10-07T06:18:18.017000
4f2c48fa-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	memory.usage	gauge	1446.0	MB	2017-10-07T06:18:18.009000
4f85c734-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.write.requests.rate	gauge	0.0684346286739	request/s	2017-10-07T06:18:18.003000
4f85c735-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.write.requests.rate	gauge	0.0	request/s	2017-10-07T06:18:18.003000
4f2c48f8-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.write.requests	cumulative	242110.0	request	2017-10-07T06:18:18.003000
4f2c48f9-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.write.requests	cumulative	0.0	request	2017-10-07T06:18:18.003000
4f2c48f7-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	memory.resident	gauge	3356.0	MB	2017-10-07T06:18:17.995000
4f2c48f6-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524	disk.write.bytes.rate	gauge	443.673279197	B/s	2017-10-07T06:18:17.991000
4f7fd89c-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.read.requests.rate	gauge	0.0	request/s	2017-10-07T06:18:17.981000
4f7fd89d-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.read.requests.rate	gauge	0.0	request/s	2017-10-07T06:18:17.981000
4f2c48f4-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.read.requests	cumulative	21020.0	request	2017-10-07T06:18:17.981000
4f2c48f5-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.read.requests	cumulative	777.0	request	2017-10-07T06:18:17.981000
4f2c48f3-ab27-11e7-85a6-00241d958c52	instance-0000000f-67e27c70-f12a-46ea-9b26-4503a3d77524-tapb21a9d82-dd	network.outgoing.bytes.rate	gauge	1.91969538594	B/s	2017-10-07T06:18:17.977000
4f2c48f1-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vda	disk.device.allocation	gauge	1932591104.0	B	2017-10-07T06:18:17.972000
4f2c48f2-ab27-11e7-85a6-00241d958c52	67e27c70-f12a-46ea-9b26-4503a3d77524-vdb	disk.device.allocation	gauge	0.0	B	2017-10-07T06:18:17.972000
4f750200-ab27-11e7-80a2-00241d958c54	67e27c70-f12a-46ea-9b26-4503a3d77524	disk.write.requests.rate	gauge	0.0684370322042	request/s	2017-10-07T06:18:17.966000

```
| 4f2c48f0-ab27-11e7-85a6-00241d958c52 | 67e27c70-f12a-46ea-9b26-4503a3d77524
| disk.write.requests              | cumulative | 242110.0          | request    | 2017-10-
07T06:18:17.966000 |
| 4f2c48ef-ab27-11e7-85a6-00241d958c52 | instance-0000000f-67e27c70-f12a-46ea-
9b26-4503a3d77524-tapb21a9d82-dd | network.incoming.packets         | cumulative |
795087.0          | packet     | 2017-10-07T06:18:17.961000 |
| 4f708fb8-ab27-11e7-80a2-00241d958c54 | instance-0000000f-67e27c70-f12a-46ea-
9b26-4503a3d77524-tapb21a9d82-dd | network.incoming.packets.rate    | gauge      |
0.210318199869    | packet/s   | 2017-10-07T06:18:17.961000 |
| 4f2c48ee-ab27-11e7-85a6-00241d958c52 | 67e27c70-f12a-46ea-9b26-4503a3d77524
| disk.allocation                  | gauge      | 1932591104.0      | B          | 2017-10-
07T06:18:17.956000 |
| 4f6b2294-ab27-11e7-80a2-00241d958c54 | 67e27c70-f12a-46ea-9b26-4503a3d77524
| disk.read.requests.rate          | gauge      | 0.0               | request/s  | 2017-10-
07T06:18:17.951000 |
+--------------------------------------+--------------------------------------------------------------------------+--
------------------------------+------------+------------------+------------+--------------------------+
```

B.2.Database setup

In order for the developed solution to work properly a Database has to be previously installed and configured, it only requires for user access configuration and to declare the table to be used by the script.

- Download and install MySQL and python packages:

  sudo apt-get install python-mysqldb mysql-server mysql-client

- Create the database and tables:

```
create database openstack;
use openstack;
    CREATE TABLE openstack_resources (
resource_id varchar(120) NOT NULL,
name varchar(40) DEFAULT NULL,
value1 varchar (20) DEFAULT NULL,
value2 varchar (20) DEFAULT NULL,
value3 varchar (20) DEFAULT NULL,
value4 varchar (20) DEFAULT NULL,
value5 varchar (20) DEFAULT NULL,
value6 varchar (20) DEFAULT NULL,
value7 varchar (20) DEFAULT NULL,
value8 varchar (20) DEFAULT NULL,
value9 varchar (20) DEFAULT NULL,
value10 varchar (20) DEFAULT NULL,
unit varchar (20) DEFAULT NULL,
time varchar(30) DEFAULT NULL,
instance_id varchar(60) DEFAULT NULL,
user_id varchar(60) DEFAULT NULL,
host_id varchar(60) DEFAULT NULL,
max_value varchar(20) DEFAULT NULL,
min_value varchar(20) DEFAULT NULL,
project_id varchar(60) DEFAULT NULL,
average varchar(20) DEFAULT NULL,
PRIMARY KEY(resource_id)
);
```

```
mysql> describe openstack_resources;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| resource_id | varchar(40) | NO   | PRI | NULL    |       |
| name        | varchar(20) | YES  |     | NULL    |       |
| value1      | varchar(20) | YES  |     | NULL    |       |
| value2      | varchar(20) | YES  |     | NULL    |       |
| value3      | varchar(20) | YES  |     | NULL    |       |
| value4      | varchar(20) | YES  |     | NULL    |       |
| value5      | varchar(20) | YES  |     | NULL    |       |
| value6      | varchar(20) | YES  |     | NULL    |       |
| value7      | varchar(20) | YES  |     | NULL    |       |
| value8      | varchar(20) | YES  |     | NULL    |       |
| value9      | varchar(20) | YES  |     | NULL    |       |
| value10     | varchar(20) | YES  |     | NULL    |       |
| unit        | varchar(20) | YES  |     | NULL    |       |
| time        | varchar(20) | YES  |     | NULL    |       |
| instance_id | varchar(20) | YES  |     | NULL    |       |
| user_id     | varchar(20) | YES  |     | NULL    |       |
| host_id     | varchar(20) | YES  |     | NULL    |       |
| max_value   | varchar(20) | YES  |     | NULL    |       |
| min_value   | varchar(20) | YES  |     | NULL    |       |
| project_id  | varchar(20) | YES  |     | NULL    |       |
| average     | varchar(20) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
21 rows in set (0,02 sec)
```

ANNEX C. DEVELOPMENTS

Three scripts were produced in this project, the first script connects to Openstack cloud, queries resource measures through API requests, parses the information and stores the results in the database; the second script is an implementation of resource assignment optimization algorithms, inputs are provided statically to the code and the algorithms implemented are FFDH, BFDH, HFF and HBF, results are printed in screen; and the third is the same as the second but taking as input a random request sequence of virtual machines. These codes can be found in the following attachments, the first one is referenced as “Update_OS_resources.py”, the second “Optim_script_tests.py” and the third “Optim_script_tests_random.py”:

Update_OS_resources.py

Optim_script_tests.py

Optim_script_tests_random.py